1-1-2001

Depth-first search embedded wavelet algorithm for hardware implementation

Li-Minn Ang
Edith Cowan University

Edith Cowan University

Copyright Warning

Depth-First Search Embedded Wavelet Algorithm for Hardware Implementation

by

Ang Li-minn, B. E. (Hons.)

Thesis submitted for the degree of

Doctor of Philosophy

School of Engineering and Mathematics

Edith Cowan University

November 2001

EDITH COWAN
UNIVERSITY
PERTH WESTERN AUSTRALIA

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Abstract

The emerging technology of image communication over wireless transmission channels requires several new challenges to be simultaneously met at the algorithm and architecture levels. At the algorithm level, desirable features include high coding performance, bit stream scalability, robustness to transmission errors and suitability for content-based coding schemes. At the architecture level, we require efficient architectures for construction of portable devices with small size and low power consumption. An important question is to ask if a single coding algorithm can be designed to meet the diverse requirements. Recently, researchers working on improving different features have converged on a set of coding schemes commonly known as embedded wavelet algorithms. Currently, these algorithms enjoy the highest coding performances reported in the literature. In addition, embedded wavelet algorithms have the natural feature of being able to meet a target bit rate precisely. Furthermore, work on improving the algorithm robustness has shown much promise. The potential of embedded wavelet techniques has been acknowledged by its inclusion in the new JPEG2000 and MPEG-4 image and video coding standards.

The consideration now is whether the algorithms can be efficiently implemented in hardware. Whereas much work has been accomplished at the algorithm level, the same cannot be said at the architecture level. The disparity between the algorithm level and the architecture level is surprising considering that we need both levels to construct portable multimedia devices. Unlike hardware architectures in general, the complexity in embedded wavelet architectures does not only lie in its computational and storage requirements. The additional complexity lies in designing tree searching schemes which can be efficiently implemented in hardware. Our focus is to design suitable tree

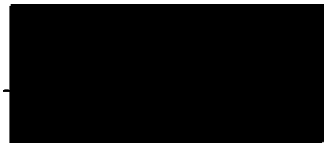
searching schemes for hardware implementation. There are basically two approaches for tree searching architectures: iterative or noniterative schemes. The noniterative scheme simplifies the tree searching but suffers from the possibility of misprediction. In this thesis, we return to the original concept of embedded wavelet algorithms using iterative tree searching. The significance of this work is to embrace iterative tree searching schemes wholeheartedly and to develop schemes which can be efficiently implemented in hardware. The iterative tree searching schemes do not suffer from misprediction.

The approach taken in this thesis is to adopt a top-down design methodology beginning from algorithm investigation and ending at hardware implementation. We begin by looking at the variations in embedded wavelet algorithms. We propose tree searching schemes based on the depth-first search (DFS). The DFS gives several advantages for hardware implementation. The DFS algorithms have low storage requirements, simple computations and allows for efficient addressing of the nodes in the tree structure. The coding performances of the DFS variants are only slightly lower than its original algorithms. Furthermore, the DFS allows for novel bit stream (BS) architectures where the coefficient magnitude bits are not stored and are processed as they flow through the architectures resulting in fast architectures with low complexity. We present a DFS BS embedded wavelet system which can be switched to perform encoding and decoding on the same device. The coding system is suitable for implementation on DSP processors, field programmable logic devices or VLSI and is a promising solution for video coding applications in general and for portable video communicators in particular. The final part of the thesis discusses the applications of the DFS algorithms for robust coding and content-based coding applications.

Statement of Originality

I declare that this thesis contains no material which has been accepted for the award of any other degree or diploma in any university. To the best of my knowledge and belief, this thesis contains no material previously published or written by any other person, except where due reference is given in the text of the thesis.

I consent to this thesis being made available for photocopying or loan.

SIGNED: 

DATE: 3/12/2001

Acknowledgments

I would like to express my gratitude to the following people at Edith Cowan University.

Prof. Kamran Eshraghian, my supervisor, for his guidance and support throughout this research project. His comments and criticism on the draft of the thesis are greatly appreciated.

My supervisor Dr. Hon Nin Cheung, for his time and advice on this project and the preparation of this thesis. His constructive feedback and suggestions are very much appreciated.

Prof. Hans-Jorg Pfleiderer, visiting professor to the School of Engineering and Mathematics in 1996, for several discussions on transform architectures and bit stream approaches.

Dr. Morteza Biglari for his help in the configuration and use of the design tools.

The assistance provided by staff and postgraduates in this department is much appreciated.

I would like to thank my parents and grandma for their unceasing love always.

List of Author's Related Publications

[1] K. Eshraghian, S. Lachowicz, G. Alagoda and L. Ang, "Architectural mappings for multimedia smart-pixel arrays," in *Proc. IEEE Int. Workshop Design, Test and Applications*, Dubrovnik, Croatia, pp. 33-35, Jun. 1998.

[2] L. Ang, H. N. Cheung and K. Eshraghian, "VLSI architecture for significance map coding of embedded zerotree wavelet coefficients," in *Proc. IEEE Asia Pacific Conf. Circuits and Systems*, Chiangmai, Thailand, pp. 627-630, Nov. 1998.

[3] L. Ang, H. N. Cheung and K. Eshraghian, "VLSI architecture for embedded zerotree wavelet coding," in *Proc. 5th Int. Symp. DSP for Communication Systems*, Perth, Australia, pp. 128-133, Feb. 1999.

[4] L. Ang, H. N. Cheung and K. Eshraghian, "VLSI decoder architecture for embedded zerotree wavelet algorithm," in *Proc. IEEE Int. Symp. Circuits and Systems*, Orlando, U.S.A, pp. 141-144, May 1999.

[5] L. Ang, H. N. Cheung and K. Eshraghian, "EZW algorithm using depth-first representation of the wavelet zerotree," in *Proc. 5th Int. Symp. Signal Processing and its Applications*, Brisbane, Australia, pp. 75-78, Aug. 1999.

[6] L. Ang, H. N. Cheung and K. Eshraghian, "Robust image compression using the depth-first search on the wavelet zerotree," in *Proc. 5th Int. Symp. Signal Processing and its Applications*, Brisbane, Australia, pp. 797-800, Aug. 1999.

[7] H. N. Cheung, G. Alagoda, K. Eshraghian and L. Ang, "Smart-pixel VLSI architecture for embedded zerotree wavelet coding," in *Proc. 5th Int. Symp. Signal Processing and its Applications*, Brisbane, Australia, pp. 693-696, Aug. 1999.

[8] L. Ang, H. N. Cheung and K. Eshraghian, "VLSI architecture for very high resolution scalable video coding using the virtual zerotree," in *Proc. IEEE Workshop Signal Processing Systems*, Taipei, Taiwan, Oct. 1999.

[9] L. Ang, H. N. Cheung and K. Eshraghian, "Comparison of winner-take-all motion compensation schemes for embedded wavelet coding," in *Proc. 6th Int. Conf. Neural Information Processing,* Perth, Australia, pp. 390-394, Nov. 1999.

[10] H. N. Cheung, L. Ang and G. Alagoda, "Bit stream architecture for the implementation of the improved embedded zerotree wavelet algorithm," in *Proc. Second Int. Conf. Information, Communications & Signal Processing,* Singapore, Dec. 1999.

[11] H. N. Cheung and L. Ang, "Analysis of embedded zerotree wavelet algorithms for robust image compression," in *Proc. Second Int. Conf. Information, Communications & Signal Processing,* Singapore, Dec. 1999.

[12] H. N. Cheung, L. Ang and K. Eshraghian, "Embedded zerotree wavelet processor for mobile video communicator," in *Proc. IEEE Int. Symp. Intelligent Signal Processing and Communication Systems,* Phuket, Thailand, Dec. 1999.

[13] H. N. Cheung, L. Ang and K. Eshraghian, "Parallel architecture for the implementation of the embedded zerotree wavelet algorithm," in *Proc. 5th Australasian Computer Architecture Conf.,* Canberra, Australia, Jan. 2000.

[14] L. Ang, H. N. Cheung and K. Eshraghian, "A dataflow-oriented VLSI architecture for a modified SPIHT algorithm using depth-first search bit stream processing," in *Proc. IEEE Int. Symp. Circuits and Systems*, Geneva, Switzerland, pp. 291-294, May 2000.

[15] H. N. Cheung and L. Ang, "Application of the EZW algorithm to content-based image compression," to be presented at *IEEE Int. Symp. Intelligent Signal Processing and Communication Systems,* Honolulu, Hawaii, Nov. 2000.

Contents

List of Figures

List of Tables

Glossary of Terms and Abbreviations

AOMSIG	All offspring marked significant
BFS	Breadth-first search
BS	Bit stream
DFS	Depth-first search
DWT	Discrete wavelet transform
EBCOT	Embedded block coding with optimized truncation algorithm
EW	Embedded wavelet
EZW	Embedded zerotree wavelet algorithm
IZ	Isolated zero
LIP	List of insignificant pixels
LIS	List of insignificant sets
LSP	List of significant pixels
MAP	Significance map
MSIG	Marked significant coefficient
NEG	Negative coefficient
NZF	Not zerotree root flag
POS	Positive coefficient
PSNR	Peak signal-to-noise ratio
SAQ	Successive approximation quantization
SCIWT	Significance checking in wavelet trees algorithm
SDF	Significant descendant flag
SGDF	Significant grand descendant flag
SIG	Significant coefficient flag
SIGN	Coefficient sign
SPIHT	Set partitioning in hierarchical trees algorithm
SP	Smart pixel
SQ	Scalar quantization
TS	Tree search
VQ	Vector quantization
WVQ	Wavelet vector quantization algorithm
ZTR	Zerotree root

Chapter 1

Introduction

The emerging technology of image communication over wireless transmission channels requires several new challenges to be simultaneously met at the algorithm and architecture levels. At the algorithm level, desirable features include high coding performance, bit stream scalability, robustness to transmission errors and suitability for content-based coding schemes. The first feature of high coding performance is essential to satisfy the limited bandwidth of wireless channels which are not currently met by conventional standards such as JPEG, MPEG and H.263. Bit stream scalability include rate scalability and spatial scalability. Rate scalability is a useful feature where the number of bits to be transmitted can be dynamically varied to suit the channel bandwidth at a particular time. Spatial scalability provides the option for multimedia applications with different screen resolutions. The third desirable feature is robustness of the transmitted bit stream over the wireless channel. Wireless communication channels suffer from burst errors in which a large number of consecutive bits are lost or corrupted by the channel fading effect. The fourth desirable feature is suitability for content-based coding schemes which is an important component in the new MPEG-4 standard. At the architecture level, we require efficient architectures for construction of portable devices with small size and low power consumption.

An important question is to ask if a single coding algorithm can be designed to meet the diverse requirements. In previous coding schemes, researchers adopted the divide and conquer approach where a single feature was targeted for improvement which was often obtained at the expense of other features. The JPEG, MPEG and H.263 standards

suffer from limited scalability features. Other schemes such as vector quantization (VQ) emphasize coding performance but require high complexity architectures. Recently, researchers working on improving different features have converged on a set of coding schemes commonly known as embedded wavelet algorithms [16], [17], [18], [19]. Currently, these algorithms enjoy the highest coding performances reported in the literature. In addition, embedded wavelet algorithms have the natural feature of being able to meet a target bit rate precisely. Furthermore, work on improving the algorithm robustness has shown much promise [26], [27], [28], [29]. In this thesis, we will also demonstrate the simplicity of incorporating content-based coding into embedded wavelet algorithms. The potential of embedded wavelet techniques has been acknowledged by its inclusion in the new JPEG2000 and MPEG-4 image and video coding standards [32].

The consideration now is whether the algorithms can be efficiently implemented in hardware. When first reported in 1993, the Embedded Zerotree Wavelet (EZW) algorithm [16] created much interest in the research community. Up till then, the coding performance of image coding algorithms which were based mostly on VQ were improved at the expense of increased complexity in the number and size of the codebooks required. The EZW algorithm reversed this trend in that it achieves high coding performance using simple scalar quantization (SQ) techniques. It achieves this because it utilizes a form of image redundancy which was not being used in other coding schemes. The algorithm reorganizes the wavelet coefficients in the subbands into a tree structure which it then efficiently encodes for transmission. The coefficients in the lowest frequency subband form the root nodes and the coefficients in the highest frequency subbands form the leave nodes. Specifically, the algorithm introduced the

concept of a zerotree. The idea is that if a coefficient in the tree is insignificant with respect to a threshold value, then there is a high probability that the descendants of the coefficient would also be insignificant. In such a case, a special symbol known as a zerotree root (ZTR) is encoded to indicate to the decoder that there are no more significant coefficients and the search can move on to another branch in the tree.

Given that the EZW algorithm uses simple SQ instead of the more complex VQ schemes, it would be natural to assume that it is suitable for hardware implementation and there would be several architectures reported in the literature. However, a search of the literature reveals that this is not the case. Whereas much work has been accomplished at the algorithm level, the same cannot be said at the architecture level. The disparity between the algorithm level and the architecture level is surprising considering that we need both levels to construct portable multimedia devices. A significant contribution at both the algorithm and architecture levels based on wavelet VQ (WVQ) has been reported very recently [33]. In this work, the authors shed some light on the complexities involved in designing embedded wavelet architectures. They postulate that embedded wavelet architectures require iterative methods to decide zerotrees. Unlike hardware architectures in general, the complexity in embedded wavelet architectures does not only lie in its computational and storage requirements. The additional complexity lies in designing tree searching schemes which can be efficiently implemented in hardware. The WVQ algorithm simplifies the tree searching requirements by using a noniterative method to decide zerotrees. The search begins at the tree root and descends to lower levels of the tree structure only if the current node is significant. There is a possibility of misprediction when the current node is

insignificant but there is a significant descendant node. The noniterative search scheme will miss this.

1.1 Significance of this Work

Embedded wavelet algorithms satisfy many desirable features for wireless image communication. The barrier impeding its potential are tree searching schemes which can be efficiently implemented in hardware. The objective of this thesis is to remove this barrier by proposing efficient embedded wavelet architectures. Our focus is to design suitable tree searching schemes for hardware implementation. There are basically two approaches for tree searching architectures: noniterative [33] or iterative schemes [34], [35], [36], [37]. The iterative schemes which have been reported makes use of memory banks to store the wavelet coefficients where all the coefficients can be accessed to establish the ancestor-descendant relationships in the tree structure. The state-of-the-art for the noniterative scheme is the WVQ algorithm [33]. The noniterative scheme simplifies the tree searching but suffers from the possibility of misprediction. In this thesis, we return to the original concept of embedded wavelet algorithms using iterative tree searching. The significance of this work is to embrace iterative tree searching schemes wholeheartedly and to develop schemes which can be efficiently implemented in hardware. The iterative tree searching schemes do not suffer from misprediction.

We observe that whereas researchers have developed several hardware techniques such as parallel processing and pipelining to handle computational complexity, comparatively fewer techniques have been developed to handle complex data structures such as lists, trees and graphs. In contrast, software techniques to handle these data

structures have been extensively developed. As the complexity of hardware systems increases, data structures become essential tools to organize data for processing and storage. Although this thesis is focused particularly on designing tree searching architectures for embedded wavelet algorithms, we hope that the design process would also provide some general insight into how to implement complex data structures in hardware.

The approach taken in this thesis is to adopt a top-down design methodology beginning from algorithm investigation and ending at hardware implementation. At each stage in the design process, we evaluate the options available and select the most suitable path for the next stage. To guide the direction of this thesis, we follow closely to the original EZW algorithm and to the Set Partitioning in Hierarchical Trees (SPIHT) algorithm [17]. The SPIHT algorithm is well known and is considered to be one of the best image coders in the literature.

1.2 Contribution of this Thesis

This thesis consists of six chapters. Other than the first and last chapters which are the Introduction and Conclusion chapters, the remaining four chapters contain contributions corresponding to the different design levels of algorithm, architecture, implementation and application.

In chapter two, we outline the variations and options available for embedded wavelet algorithms and discuss the variations which are suitable for hardware implementation. We present a variation based on depth-first search (DFS) processing which has reduced storage and simpler processing for hardware implementation. The coding performances of the DFS variants are only slightly lower than its original embedded wavelet

algorithms and are higher than the WVQ algorithm at high bit rates and is comparable to the WVQ algorithm at lower bit rates.

Chapter three presents new architectures to implement embedded wavelet algorithms based on the DFS representation of the coefficient tree. We consider two issues. The first issue is the simplicity of the architectures for hardware implementation and the second issue is the flexibility of the architectures to implement variations in the algorithms. For the first issue of simplicity, we present novel bit stream (BS) architectures which processes the bits as they flow through the processor. For the second issue of flexibility, we present parallel DFS BS architectures which provides options to output in different formats to achieve higher coding performances.

Chapter four presents hardware approaches for the implementation of the DFS BS SPIHT system. The input into the system during encoding is a stream of image pixels and the output is an encoded bit stream ready for transmission. The system produces the reconstructed image data from the received bit stream during decoding. We present a hardware implementation of the DFS BS SPIHT system which can be switched to perform encoding and decoding on the same device.

Chapter five presents applications using the DFS embedded wavelet algorithms. We will discuss two applications for the DFS algorithms. The first application is the use of the DFS EZW algorithm for robust coding and the second application is for content-based coding using the DFS implementation of the Improved EZW algorithm.

Chapter 2

Embedded Wavelet Algorithms for Hardware Implementation

2.1 Introduction

A study into the different variations available for embedded wavelet (EW) algorithms may provide two benefits. Firstly, it may allow the development of an algorithm variation which has a higher coding performance than existing algorithms. And secondly, a variation of the algorithm may be found to be suitable for hardware implementation. The study of EW algorithms which are suitable for hardware implementation lags behind the study to improve the performance of the algorithms which is very far advanced. With the introduction of the JPEG2000 and MPEG-4 coding standards, there has been increasing interest in implementing EW algorithms in hardware. Previous approaches to simplify EW algorithms for hardware implementation have looked primarily at reducing the storage requirements [40], [41], [42], [43]. Other factors which are desirable for hardware implementation are simpler processing and efficient addressing of wavelet coefficients in the tree structure. The first two factors of lower storage requirements and simpler processing apply to hardware architectures in general. The third factor of efficient addressing of nodes in the tree structure is unique to the implementation of tree-based coding schemes such as EW algorithms.

This chapter is organized as follows. In Section 2.2, we outline the variations and options available for EW algorithms. In Section 2.3, we present a tree searching (TS) variation based on depth-first search (DFS) processing which allows for efficient

addressing of coefficient nodes in the tree structure. In addition, the DFS variants require less storage than previous approaches and allows for simpler processing. We perform simulations to compare the coding performances of the DFS variants with its original EW algorithms in Section 2.4 and present conclusions in Section 2.5.

2.2 Embedded Wavelet Algorithms and its Variations

Figure 2.1 shows a block diagram of an EW algorithm. The discrete wavelet transform (DWT) transforms the image pixels into its corresponding wavelet coefficients. These wavelet coefficients are passed into the TS module to generate a symbol stream. The generated symbol stream is then passed into an arithmetic coder to obtain a compressed bit stream which is transmitted.

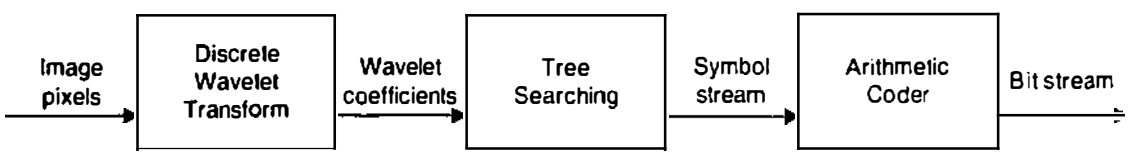


Figure 2.1 Block diagram of an embedded wavelet algorithm

The symbol stream is generated from multiple passes of coding the coefficients where a threshold value is associated with each pass. Initially, the threshold value is set to half the maximum value of the coefficients and is further halved for each subsequent pass. The number of passes to be performed depends on the target bit rate to be met. Two types of symbols are contained in the symbol stream:

- Significance map (MAP) symbols which encode the positions and signs of the significant coefficients in the tree structure and

- Successive-approximation quantization (SAQ) symbols which encode the approximate magnitudes of the significant coefficients in decreasing order of importance.

A coefficient is defined as significant if the magnitude of the coefficient value is greater than or equal to the threshold value. The order of generation of the symbol stream alternates between MAP and SAQ symbols as shown in Figure 2.2.

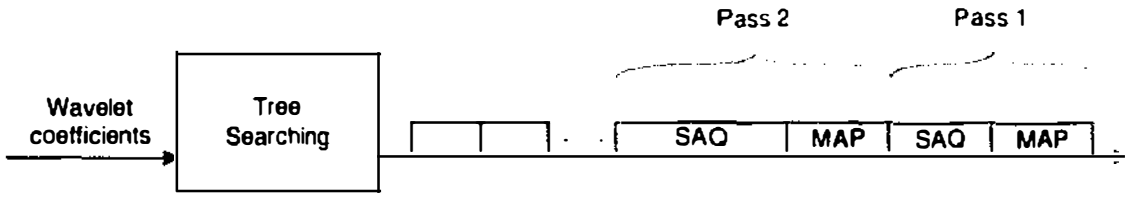


Figure 2.2 Tree searching module

The feature common in all EW algorithms is that the algorithms establish a tree structure amongst the coefficients in the subbands. The algorithms then use the tree structure to efficiently encode the MAP. But the algorithms differ in:

- The structure of the coefficient tree used;
- The strategy to search the coefficient tree for significant coefficients;
- The scheme to code the coefficient tree and the
- Generation and transmission of the symbol stream so that symbols which are more significant are transmitted ahead of less significant symbols.

The fourth difference of generation and transmission of the symbol stream applies not only to MAP coding but also to SAQ coding where the SAQ symbols can be reordered for transmission as in the EZW algorithm.

2.2.1 Coefficient Tree Structure

The first variation available for EW algorithms is the structure of the coefficient tree. Several EW algorithms use the tree structure in the EZW algorithm. Figure 2.3 shows a three scale DWT decomposition and the tree structure for the EZW algorithm. From the lowest frequency subband, the coefficients begin an ancestor-descendant hierarchy of coefficients descending all the way to the highest frequency subbands. Each parent coefficient in the tree structure has four children coefficients except at the lowest frequency subband where each parent coefficient has three children coefficients. The coefficient tree structure contains a number of subtrees equal to the number of coefficients in the lowest frequency subband.

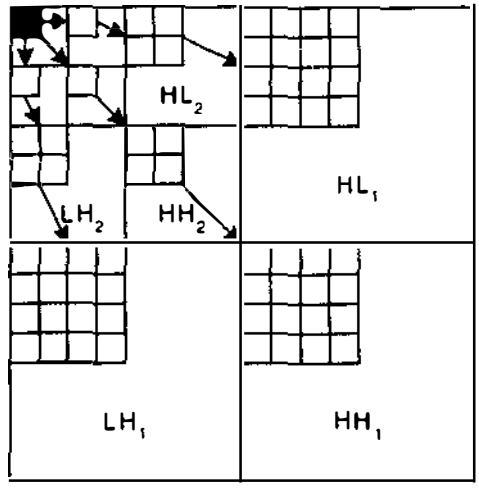


Figure 2.3 Ancestor-descendant relationship of subbands in EZW algorithm

Figure 2.4 shows the structure of a coefficient subtree. The subtree is ordered in several ways:

- Firstly, the tree is ordered from the lowest frequency subbands at the root node to the highest frequency subbands at the leaf nodes;

- Secondly, with the exception of the lowest frequency subband, the three branches of the tree are ordered according to their frequency subbands and
- Thirdly, the root node links the three frequency branches according to their spatial position in the subbands. The frequency branches are arranged in the HL, LH and HH order.

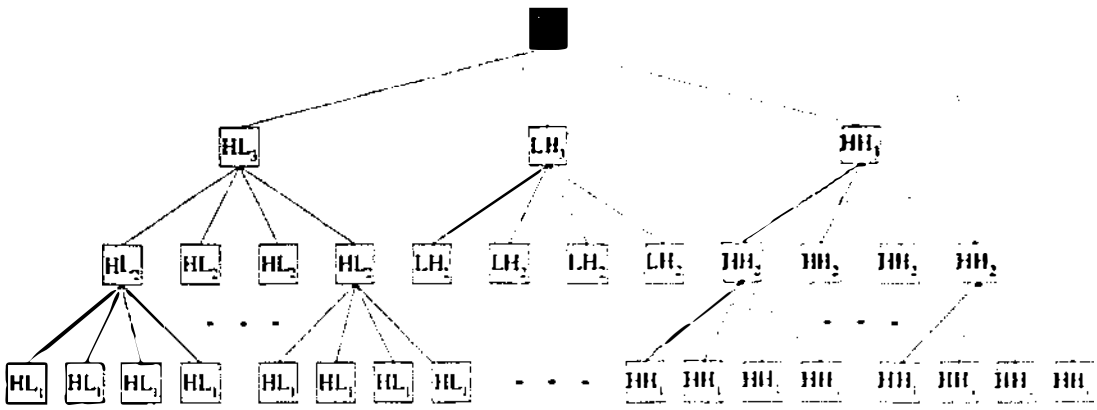


Figure 2.4 EZW coefficient subtree

A variation of the coefficient tree structure is that used in the SPIHT algorithm. One notable difference between the tree structure in the SPIHT algorithm and the tree structure in the EZW algorithm is in the linking of the root nodes in the lowest frequency subband. In the EZW tree structure, each coefficient in the lowest frequency subband is linked to three children nodes in the next higher frequency subbands. In the SPIHT tree structure, each coefficient in the lowest frequency subband is either linked to four children nodes or to none. Figure 2.5 shows the ancestor-descendant relationship for the SPIHT algorithm. The root node marked by the asterisk has no descendants and is not linked to the three children nodes.

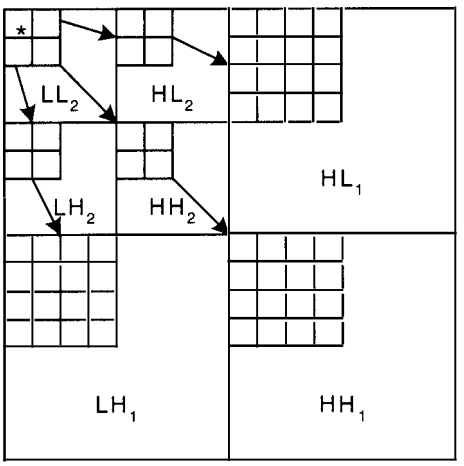


Figure 2.5 Ancestor-descendant relationship of subbands in SPIHT algorithm

Figure 2.6 shows the coefficient subtree for the SPIHT algorithm. A point to note is that the tree structure is for a two scale DWT decomposition. The inconsistent treatment of the root nodes in the SPIHT tree structure increases the complexity for hardware implementation.

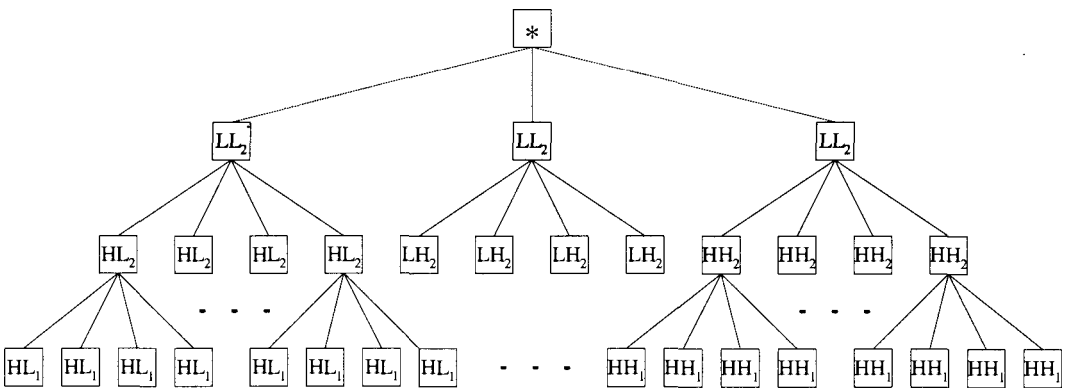
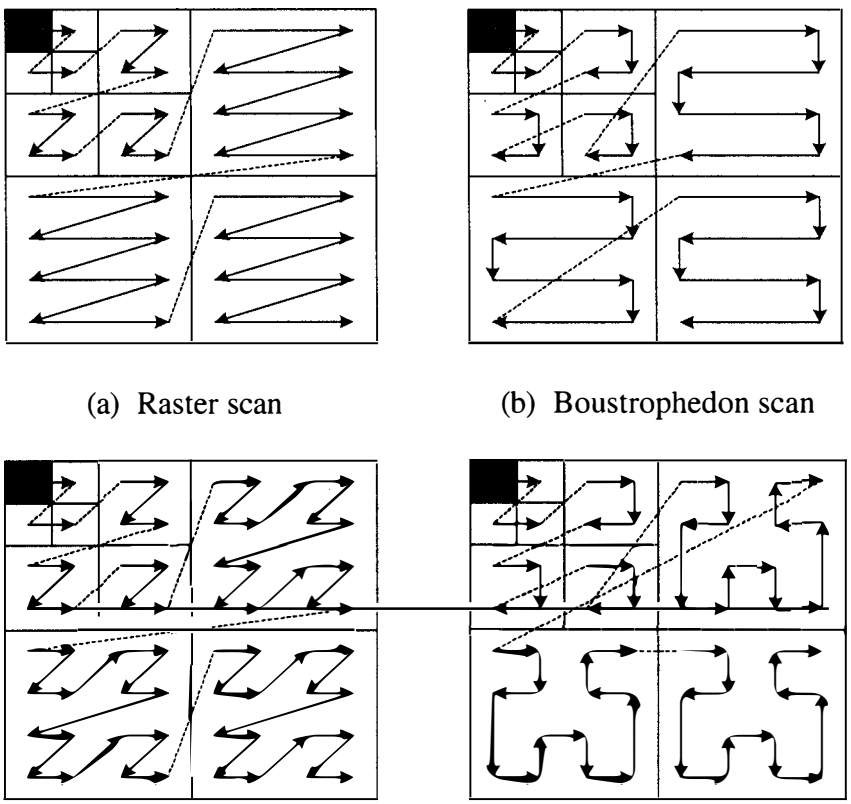


Figure 2.6 SPIHT coefficient subtree

2.2.2 Tree Searching Scheme

A second variation amongst EW algorithms is in the scheme to search the coefficient tree for significant coefficients. Figure 2.7 shows the different schemes which can be used to scan the wavelet subbands. The simplest scheme for hardware implementation

is the raster scan. The EZW algorithm uses the raster scan scheme. The scan begins from the lowest frequency subband and after all the coefficients in the subband have been scanned proceeds to the next higher frequency subband. A subband scanning scheme with slightly more complexity is the boustrophedon scan from the Greek for "how an oxen plows a field". This scan is similar to the raster scan except that the scan order for every other row is reversed. The third subband scanning scheme which can be used is the Morton scan. This scanning scheme is similar to the raster scan except that the coefficients in the subbands are scanned in a fractal-like manner. The final subband scanning scheme with the highest complexity for hardware implementation is the peano scan. This scan combines the row reversal of the boustrophedon scan with the fractal manner of the Morton scan.



(a) Raster scan (b) Boustrophedon scan

(c) Morton scan (d) Peano scan

Figure 2.7 Subband scanning schemes

The four subband scanning schemes in Figure 2.7 are similar in that the schemes scan all the coefficients in the lowest frequency subband first before scanning the coefficients in the higher frequency subbands. The root nodes in the coefficient tree structure are scanned first. The subband scanning schemes are similar to the breadth-first search (BFS). BFS traversal scans the tree structure at different levels beginning from the root nodes of the tree and ending at the leaf nodes. The scanning order for the tree structure in Figure 2.4 is LL_3, HL_3, LH_3, HH_3, HL_2, HL_2, HL_2, HL_2, LH_2, ... , HH_1, HH_1, HH_1, HH_1. For a coefficient tree with more than one subtree, BFS scans all the nodes on the same level before it scans the nodes in another level. Figure 2.8 shows the BFS traversal of a coefficient tree containing N subtrees.

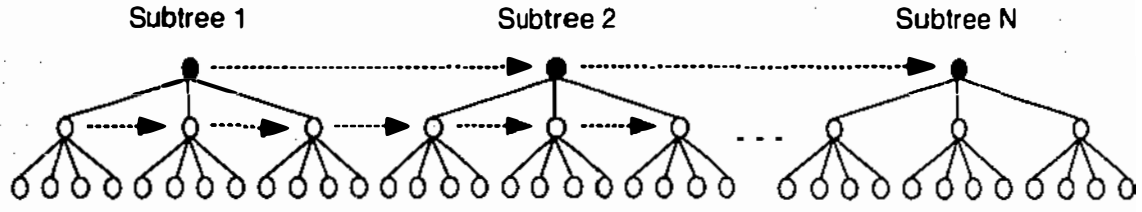


Figure 2.8 BFS traversal of coefficient tree containing N subtrees

In contrast to the subband scanning schemes and the BFS, depth-first search (DFS) traversal scans the tree structure beginning from the root node and descends all the way to the leaf nodes before it backtracks up a level. The scanning order for the tree structure in Figure 2.4 is LL_3, HL_3, HL_2, HL_1, HL_1, HL_1, HL_1, HL_2, ... , HH_2, HH_1, HH_1, HH_1, HH_1. For a coefficient tree with more than one subtree, DFS scans all the nodes in a subtree before it scans the nodes in another subtree. The DFS performs a natural partitioning of the coefficient tree into a number of subtrees which can be processed independently. Figure 2.9 shows the DFS traversal of a coefficient tree containing N subtrees.

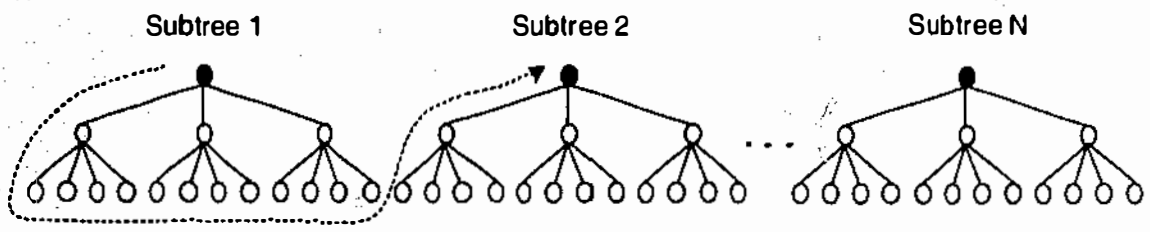


Figure 2.9 DFS traversal of coefficient tree containing N subtrees

The search order for the BFS and DFS tree traversal schemes are fixed and the traversal follows a predetermined order. An example of a TS strategy where the traversal is not fixed is the search strategy used in the SPIHT algorithm. In this algorithm, three ordered lists are maintained to store the significance information. The three lists are the list of insignificant sets (LIS), list of insignificant pixels (LIP) and list of significant pixels (LSP). The search order of the coefficient tree is according to the set partitioning strategy. The coefficients are processed in the order indicated in the LIS.

2.2.3 Tree Coding Scheme

Another variation for EW algorithms is the scheme used for MAP coding of the coefficient tree. SAQ coding is similar in all EW algorithms. For the coding of the MAP in each pass, each node in the coefficient tree is associated with a set of binary symbols as shown in Table 2.1. The MAP symbols are divided into two categories: coding symbols and processing symbols.

Table 2.1 MAP symbols for coding of coefficient tree

Category	Symbol	Description
Coding	SIGN	Sign bit of the coefficient
	SIG	Significant Coefficient Flag - indicate if this coefficient position is significant
	SDF	Significant Descendant Flag - indicate if this coefficient position has any significant descendant coefficient
	SGDF	Significant Grand Descendant Flag - indicate if this coefficient position has any significant descendant coefficient with the exception of its immediate children
Processing	MSIG	Marked Significant Coefficient Flag - indicate if this coefficient has already been found to be significant in a previous pass
	AOMSIG	All Offspring Marked Significant Coefficient Flag - indicate that all the descendants of the coefficient are already MSIG nodes

MAP coding symbols are the only symbols sent to the decoder. The SDF and SGDF symbols are only sent to the decoder in cases when the decoder cannot infer their values. For example, the SDF symbol is not sent for leaf nodes when the decoder can infer that SDF = 0. Similarly, the SGDF symbol is not sent for leaf nodes and their parent nodes as the decoder can infer that SGDF = 0. The SDF and SGDF symbols code the significance of the descendants. In effect, these symbols are used to tell the decoder how to search the coefficient tree. Table 2.2 shows the effect on the tree search using these symbols.

Table 2.2 Effect on tree search using SDF and SGDF symbols

SDF	SGDF	Effect on tree search
0	X	Stop search after current node
1	0	Stop search after children nodes
1	1	Continue search

Figure 2.10 shows an example of a coefficient subtree and Table 2.3 shows the MAP coding symbols for a threshold value of 32. In the table, the root coefficient -31 is less than the threshold value and the SIG symbol is set to 0. The coefficient value is

negative which is indicated by the SIGN symbol of 1. The coefficient has at least one significant descendant coefficient and the SDF symbol is set to 1. The coefficient has a significant grand descendant and the SGDF symbol is set to 1.

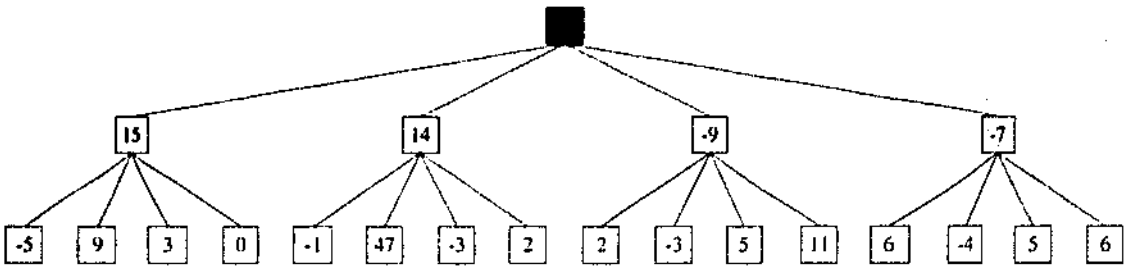


Figure 2.10 Example of coefficient subtree

Table 2.3 Example of MAP coding symbols for a threshold value of 32

Coefficient	SIG	SIGN	SDF	SGDF
-31	0	1	1	1
15	0	0	0	0
-5	0	1	0	0
9	0	0	0	0
3	0	0	0	0
0	0	0	0	0
14	0	0	1	0
-1	0	1	0	0
47	1	0	0	0
-3	0	1	0	0
2	0	0	0	0
-9	0	1	0	0
2	0	0	0	0
-3	0	1	0	0
5	0	0	0	0
11	0	0	0	0
-7	0	1	0	0
6	0	0	0	0
-4	0	1	0	0
5	0	0	0	0
6	0	0	0	0

MAP processing symbols are not sent to the decoder but are only kept at the encoder and decoder for processing. The MSIG symbol denotes which coefficients have already

been found to be significant in previous passes. Other than the MSIG symhol, an additional symbol which can be kept at the encoder and decoder for processing is the AOMSIG symbol which is used to indicate that all the descendants of the coefficient are already MSIG nodes. When AOMSIG = I, the encoder does not need to send the SDF and SGDF symbols as the decoder can infer that SDF = 0 and SGDF = 0.

The MAP symbols in Table 2.1 can be used to implement tree coding schemes for a number of EW algorithms. For example, the MAP coding representation for the EZW algorithm is shown in Table 2.4. The symbol values in italics denote that this information is used for encoding the symbols only and is not part of the symbols. It is "assumed" and therefore it is not sent. In the table, five MAP symbols are used to represent the various information of the coefficient tree in the current pass. In terms of the search, the isolated zero (IZ) symbol explicitly indicates that the search should continue down to lower levels from this coefficient while the zerotree root (ZTR) symbol indicates that the search should stop at this coefficient level.

Table 2.4 MAP coding scheme for EZW algorithm

	Symbol	SIG	SIGN	SDF	SGDF
MSIG = 0	POS	1	0	*1*	*1*
	NEG	1	1	*1*	*1*
	IZ	0	X	1	*1*
	ZTR	0	X	0	*0*
	Z	0	X	*0*	*0*
MSIG = 1	MSIG	*0*	X	*1*	*1*

For the positive (POS) and negative (NEG) symbols, the original EZW algorithm assumes that if the current coefficient is significant, then there would probably be significant coefficients which are descendants of the current coefficient at lower levels. Based on this assumption, the search will continue to lower levels. However, if there

are no significant descendant coefficients, additional ZTR symbols would be required at lower levels to stop the search. The information to indicate whether the search should continue from a node to lower levels below is by the SDF symbol. In the case of the IZ symbol, SDF = 1, while for the ZTR symbol, SDF = 0. For MSIG coefficients, the original EZW algorithm continues the search to lower levels.

To incorporate explicitly the information to the POS and NEG symbols if the search should continue to lower levels or not from the corresponding coefficient node. we attach the SDF to these two symbols. Shown in Table 2.5 are the modified POS and NEG symbols together with other modifications that improve the performance of the EZW algorithm. Some of the modifications are similar to those reported in [19] but we describe the modifications in relation to the MAP coding representation.

Table 2.5 MAP coding scheme for Improved EZW algorithm

	Symbol	SIG	SIGN	SDF	SGDF
MSIG = 0	POS_SD	1	0	1	*1*
	POS_ZT	1	0	0	*0*
	POS_LEAF	1	0	*0*	*0*
	NEG_SD	1	1	1	*1*
	NEG_ZT	1	1	0	*0*
	NEG_LEAF	1	1	*0*	*0*
	IZ	0	X	1	*1*
	ZTR	0	X	0	*0*
	ZTR_LEAF	0	X	*0*	*0*
MSIG = 1	IZ_MSIG	*0*	X	1	*1*
	ZTR_MSIG	*0*	X	0	*0*

While the first two bits, SIG and SIGN are the same as those in Table 2.4 for the POS and NEG symbols, the differences are in the additional SDF bit. For the POS_SD (Positive Significant Coefficient with Significant Descendant), the SDF has a value of 1, while for the POS_ZT (Positive Significant Coefficient and Zerotree Root), the value is 0. The binary representations of these two symbols are 101 and 100 respectively. The

other symbol POS_LEAF (Positive Significant at Leaf Node) is for the case that the coefficient is at the lowest level of the tree, i.e. a leaf node, where there are no descendants. In this case, SDF is always 0 and is not sent, and the binary representation for this symbol is 10. The corresponding symbols for negative significant coefficients are shown in the table.

The other symbols are for the cases where the coefficients are not significant or the coefficients have been found significant in the previous passes. In the latter case, SIG is always 0. The symbols IZ and ZTR have the same meanings as those shown in Table 2.4. The new symbol IZ_MSIG indicates that the coefficient has been found significant in a previous pass. Therefore the SIG bit is not sent since we do not need to send any more significant information about this coefficient. Similarly, there is no need to send the SIG bit for ZTR_MSIG. The ZTR_LEAF symbol in Table 2.4 indicates a non-significant coefficient at a leaf node where only the SIG = 0 needs to be sent. When a coefficient at a leaf node is found significant, it will have MSIG set to 1 and SIG to 0 in all subsequent passes. Together with the fact that SDF is always 0 for a leaf node, a MSIG leaf node is virtually deleted from the coefficient tree for future passes of the search. The other case, which will not happen, is that a leaf coefficient will never have an IZ symbol.

Two kinds of modifications have been made to the basic EZW algorithm. The first one is to attach an additional bit, i.e. SDF, to the significant coefficient symbols. This will increase the number of bits to code a significant coefficient by 1 in some cases, but will decrease the number of ZTR symbols if the corresponding coefficient has no significant descendants. Whether this will lead to an improved performance of the EZW algorithm depends on the characteristics of the image being coded [19]. The other

kind of modifications are certain to reduce the number of bits required to code an image by not sending the information the decoder already has. The MAP coding representation for the Improved EZW algorithm can be further improved by explicitly sending the SGDF bit. The SGDF bit is only sent if SDF = 1. This scheme would be similar to the MAP coding scheme for the SPIHT algorithm.

2.2.4 Generation and Transmission of Symbol Stream

The fourth variation available for EW algorithms is the generation and transmission of the symbol stream. For high performance, symbols which are more significant should be transmitted ahead of symbols which are less significant. In each coding pass, the SAQ symbols should be transmitted ahead of the corresponding MAP symbols. In the EZW and SPIHT algorithms, coefficients which have been found to be significant (MSIG) are taken out of the coefficient tree and put into a separate queue. During each pass, the coding of the MAP symbols are generated from traversing the coefficient tree and the SAQ symbols are generated from scanning the MSIG coefficient queue. The order of generation of the symbol stream alternates between MAP and SAQ symbols.

A variation for the generation of the symbol stream would be to leave MSIG coefficients in the coefficient tree. When traversing the coefficient tree, when MSIG = 0 a MAP symbol is generated and when MSIG = 1, a SAQ symbol is generated. The symbol stream would be a mixture of MAP and SAQ symbols as shown in Figure 2.11. Another option available for EW algorithms is to reorder the SAQ symbols before transmission. The SAQ symbols are reordered in descending order based on the reconstructed magnitude of the wavelet coefficients at the decoder as in the EZW algorithm.

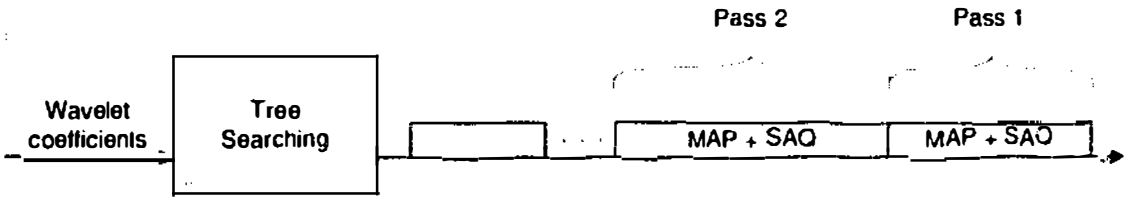


Figure 2.11 Mixed MAP and SAQ symbol stream

2.3 Features of DFS for Hardware Implementation

In this section, we present a variation of EW algorithms for hardware implementation based on depth-first search (DFS) processing. The features of the DFS EW algorithms are:

1. The depth-first tree searching strategy is used;
2. Coefficients which have been found to be significant are not taken out of the coefficient tree but are marked significant (MSIG);
3. The symbol stream is a mixture of MAP and SAQ symbols;
4. The SAQ symbols are not reordered for transmission;
5. The wavelet coefficients are processed using the sign-magnitude representation and
6. The threshold values are powers of two.

Figure 2.12 shows the DFS variant for the EZW algorithm. The DFS variants for the Improved EZW and SPIHT algorithms have similar structure. The wavelet coefficients are in the sign-magnitude representation where the sign bits are stored in SIGN and the magnitude bits are stored in MAG. The number of magnitude bits used to represent the coefficients is given by n and each tree node is represented by (i). The symbol LEAF(i) indicates if the node is a leaf in the coefficient tree. The DFS algorithms have advantages for hardware implementation. They have reduced storage, simpler computation and allows for efficient addressing of the nodes in the coefficient tree.

DFS EZW Algorithm:

1) **Initialization:** Output n and set MSIG(i) to 0 for all tree nodes

2) **Coding Pass:**

For each node (i) in the coefficient tree do:
If MSIG(i) = 1 then
- Output MAG(i,n)

Else
- Output SIG(i)

If SIG(i) = 1 then
- Output SIGN(i)
- Set MSIG(i) = 1

Else
If LEAF(i) = 0 then
If AOMSIG(i) = 0 then
- Output SDF(i)

End If
End If
End If
End If
End For

3) **Quantization-Step Update:** decrement n by 1 and go to Step 2

Figure 2.12 DFS EZW algorithm

2.3.1 Partitioning of Coefficient Tree into Subtrees

Feature (1) is that the depth-first tree searching strategy is used. The DFS performs a natural partitioning of the coefficient tree into subtrees which can be processed independently. Figure 2.13 shows the storage arrangements for the BFS and DFS tree traversal schemes. The BFS arrangement is shown in Figure 2.13(a). The root nodes in the coefficient tree are located at level 0 and the leaf nodes are located at level 2. The BFS arrangement stores the coefficient tree according to its different levels. By scanning the storage in a sequential manner, all the root nodes at level 0 will be processed before the nodes at the next level are processed. Figure 2.13(b) shows the

DFS arrangement. The DFS performs a natural partitioning of the coefficient tree into a number of subtrees which are arranged one after another in the storage. The storage requirements can be reduced to the storage required for a single subtree. After one subtree has been processed, the storage can be reused for processing the next subtree.

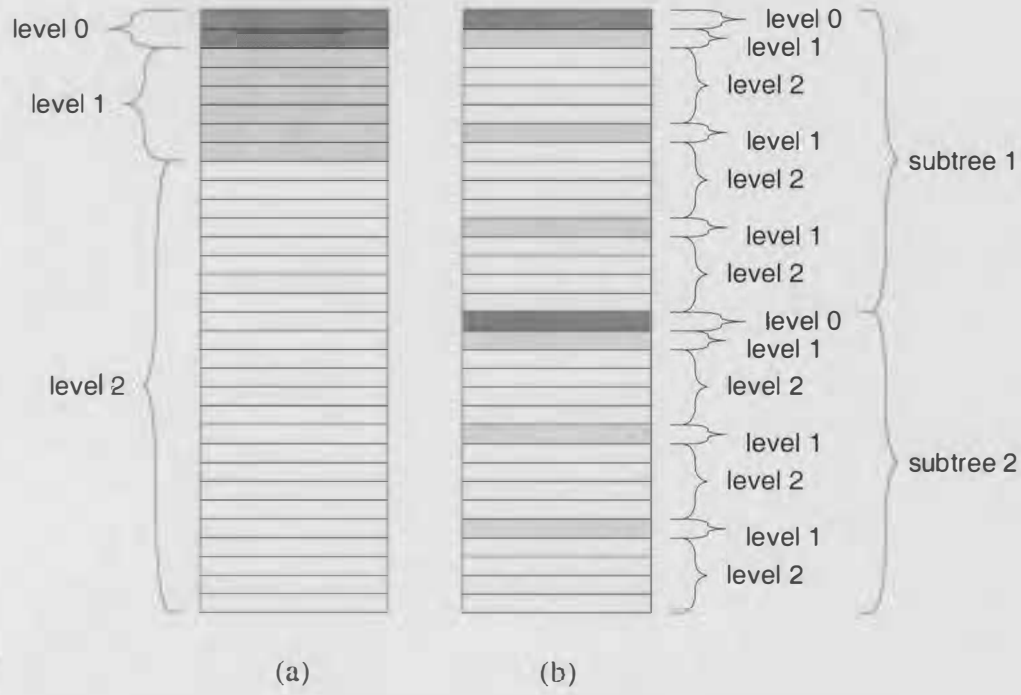


Figure 2.13 DFS partitioning of coefficient tree into subtrees

2.3.2 Propagation of Significance Symbols

One complexity for processing of the coefficient tree is the determination of the SDF and SGDF significance symbols. The DFS simplifies the propagation of the significance symbols in the tree by propagating the significance symbols to the parent node as soon as the necessary children nodes have been searched. Figure 2.14 shows the propagation order of the significance symbols using the DFS. The propagation order begins from the leaf nodes and ends at the root node. The DFS scan begins from the HH_1 children nodes and as soon as the four children nodes have been searched, the

significance symbols are propagated to the HH_2 parent node. After the HH_2, LH_2 and HL_2 nodes have been searched, the symbol is propagated to the LL_2 node. The significance symbols for the LL_2 root node can be immediately determined after the determination of the significance symbols for the HH_2, LH_2 and HL_2 branch nodes.

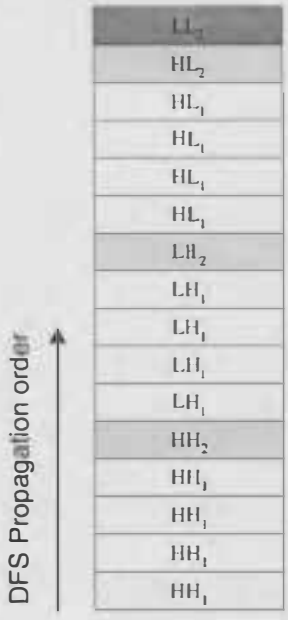


Figure 2.14 DFS propagation order

2.3.3 MSIG Quantization

In EW algorithms, the coding of the MAP symbols are generated from traversing the coefficient tree and the SAQ symbols are generated from scanning the MSIG coefficient queue. In each coding pass, coefficients which have been found to be significant are transferred from the coefficient tree to the MSIG coefficient queue. This would require variable storage schemes or a worst-case of allocating the maximum storage required for both the coefficient tree and MSIG coefficient queue. Feature (2) removes the need for a separate storage for MSIG coefficients. The MSIG symbol is used to keep track of which coefficients have been found to be significant. Similar approaches to reduce

storage requirements for coefficients which have been found to be significant have been used in [40], [41], [42]. These approaches use the set partitioning scheme for the search strategy. The DFS approach requires less storage because it is a fixed traversal scheme. For a 512×512 grayscale image, the set partitioning approach requires 320K of storage while the DFS approach requires 256K of storage.

2.3.4 Bit Stream Processing

Feature (3) is to transmit the symbols as they are generated to avoid having to store the symbols before they are transmitted. Feature (4) omits the high processing cost of reordering the SAQ symbols for transmission. Features (5) and (6) use the advantages of bit stream processing using powers of two threshold values. They take advantage of the fact that the wavelet coefficients are in their sign-magnitude representation. One requirement during encoding is to determine the SIG symbol. In BS processing, this can be directly obtained from the magnitude representation of the coefficient without having to perform any comparison operations. For example, in the DFS EZW algorithm in Figure 2.12. SIG(i) can be simply replaced with MAG(i,n). In decoding, one requirement is the calculation of the approximated coefficient magnitudes after the end of transmission. Using bit stream processing, this can be easily achieved by treating the MSIG symbol as if it is the last SAQ symbol received from the encoder. This has the effect of appending a '1' to the approximated magnitudes of MSIG nodes. For nodes which are not MSIG, the values of the approximated magnitudes remain 0. The approximated magnitudes can be determined without having to perform any arithmetic operations.

2.4 Simulation Results

2.4.1 Comparison of TS Strategies

For an initial investigation, the BFS and DFS tree searching strategies were simulated using the Lena and Barbara 512×512 images and the peak signal-to-noise ratio (PSNR) (dB) for the images were calculated. Figure 2.15 and Figure 2.16 show the Lena and Barbara test images respectively. The simulations were written to follow the EZW algorithm as closely as possible and only the tree searching strategies were changed. We used the wavelet filters and adaptive arithmetic coder used in the EZW algorithm as described in [46] and [47] respectively.

Figure 2.17 and Figure 2.18 show the coding performances which were obtained for the Lena and Barbara images respectively for various bits per pixel (bpp). The performance for the EZW algorithm is shown for comparison. The BFS gave similar results to EZW. This is expected considering that the two search strategies are similar. The DFS results in a slight decrease in performance when compared with EZW and BFS. For the Lena image at 0.25 bpp, EZW gives a PSNR of 33.17 dB. The PSNR obtained using BFS and DFS were 33.28 dB and 32.81 dB respectively. There is a reduction of about 0.5 dB using DFS.

Figure 2.15 Lena test image

Figure 2.16 Barbara test image

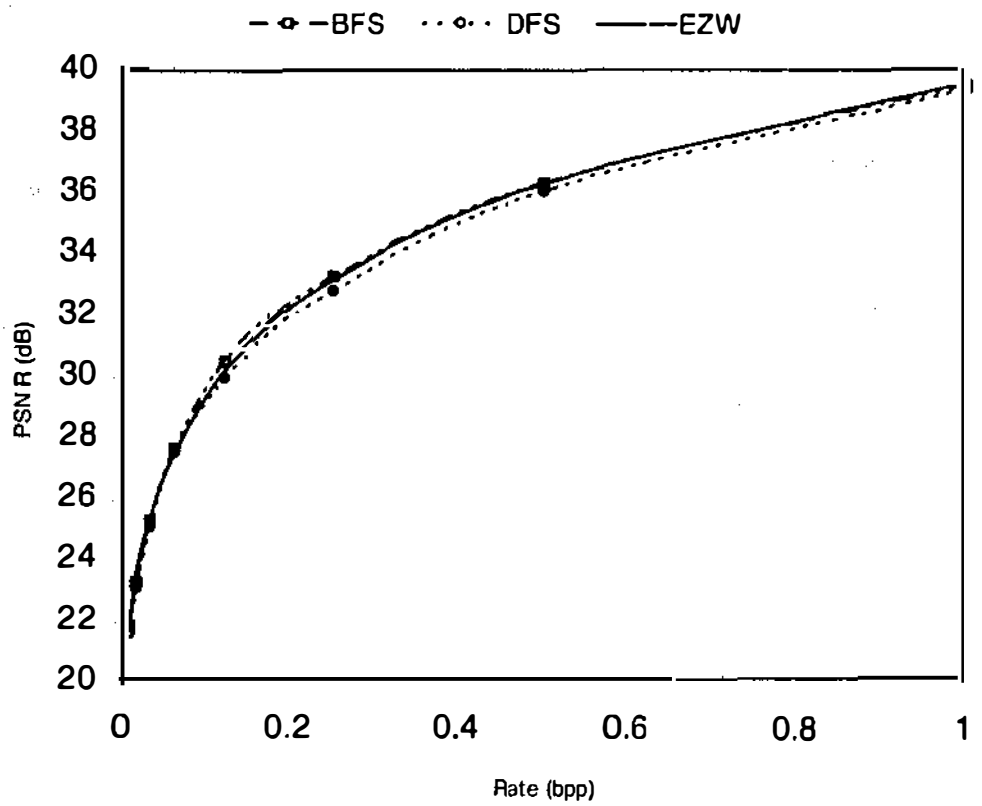


Figure 2.17 Simulation results for TS strategies for Lena image

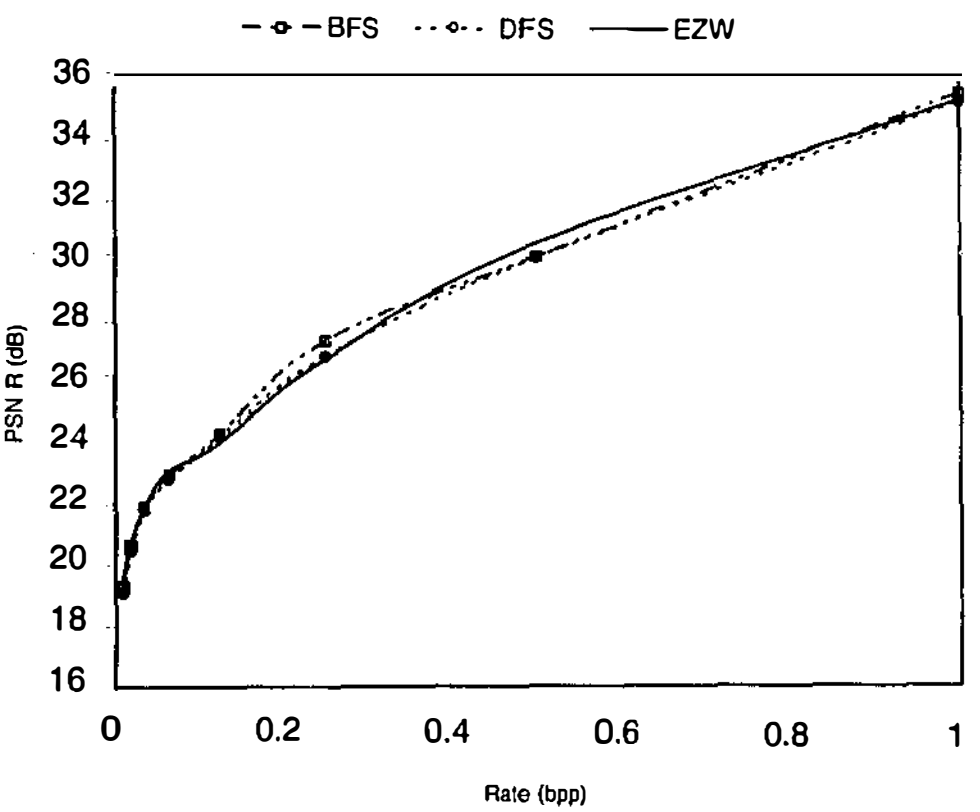


Figure 2.18 Simulation results for TS strategies for Barbara image

2.4.2 Comparison of DFS Configurations

We then performed simulations on the following simplified configurations of the EZW algorithm using the DFS tree searching strategy as shown in Table 2.6. The four configurations were applied to the Lena and Barbara images. Configurations 1 and 2 gave similar results, as did Configurations 3 and 4 showing that reordering of SAQ symbols does not significantly affect the performance. A similar conclusion about SAQ reordering has been reported in [48] for the raster scan used in the EZW algorithm. Figure 2.19 and Figure 2.20 show the coding performances versus bit rates for Configurations 1 and 4 for the Lena and Barbara images respectively. The average difference between the PSNRs produced by Configurations 1 and 4 is 2.2 dB showing that arithmetic coding does give a significant improvement in the DFS EZW algorithm.

Table 2.6 DFS Configurations

Configuration	Description
1	This configuration consists of the DFS arrangement. The SAQ symbols are not reordered and the output bit stream is not arithmetically coded
2	It consists of the DFS arrangement and the SAQ symbols are reordered. It is expected that the performance of this configuration will be the same as that of Configuration 1 if all the information resulting from a pass of the scanning of the coefficient tree is completely transmitted to the decoder. This configuration produces better results if the transmission is truncated during the sending of the SAQ
3	This configuration consists of the DFS arrangement and the arithmetic coder (AC). This configuration is expected to perform better than Configuration 1 due to the further compaction of the output bit stream by the AC
4	It consists of all the components of the EZW algorithm using DFS. This configuration will produce the best performance compared to the other configurations

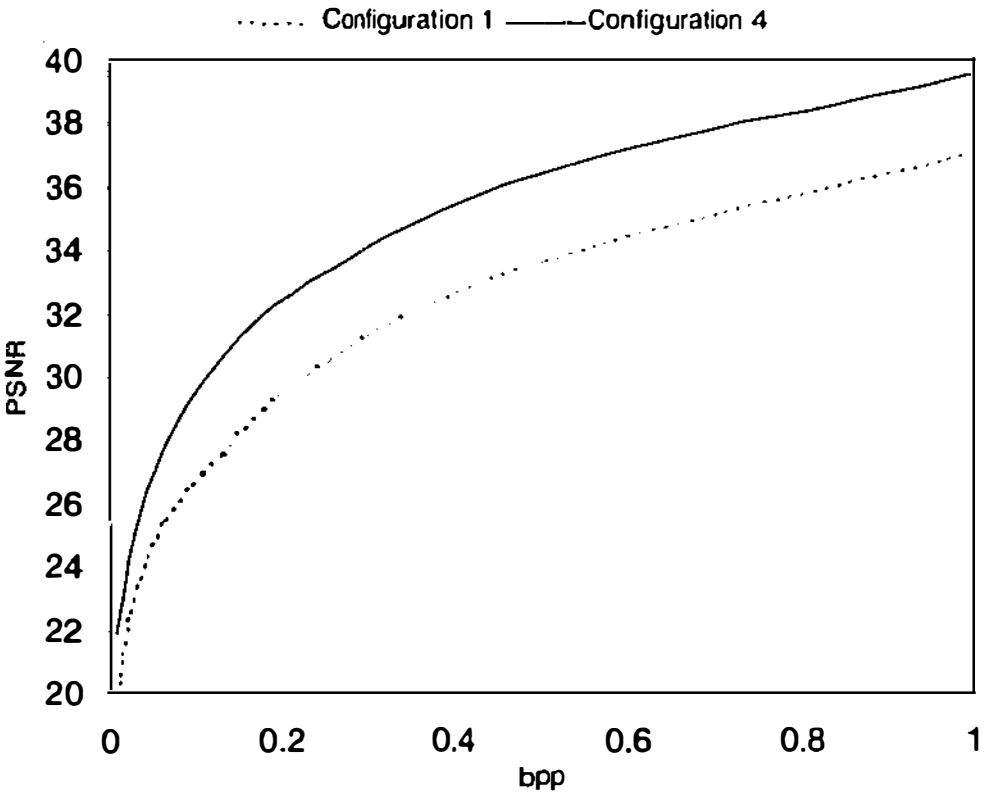


Figure 2.19 Simulation results for DFS Configurations for Lena image

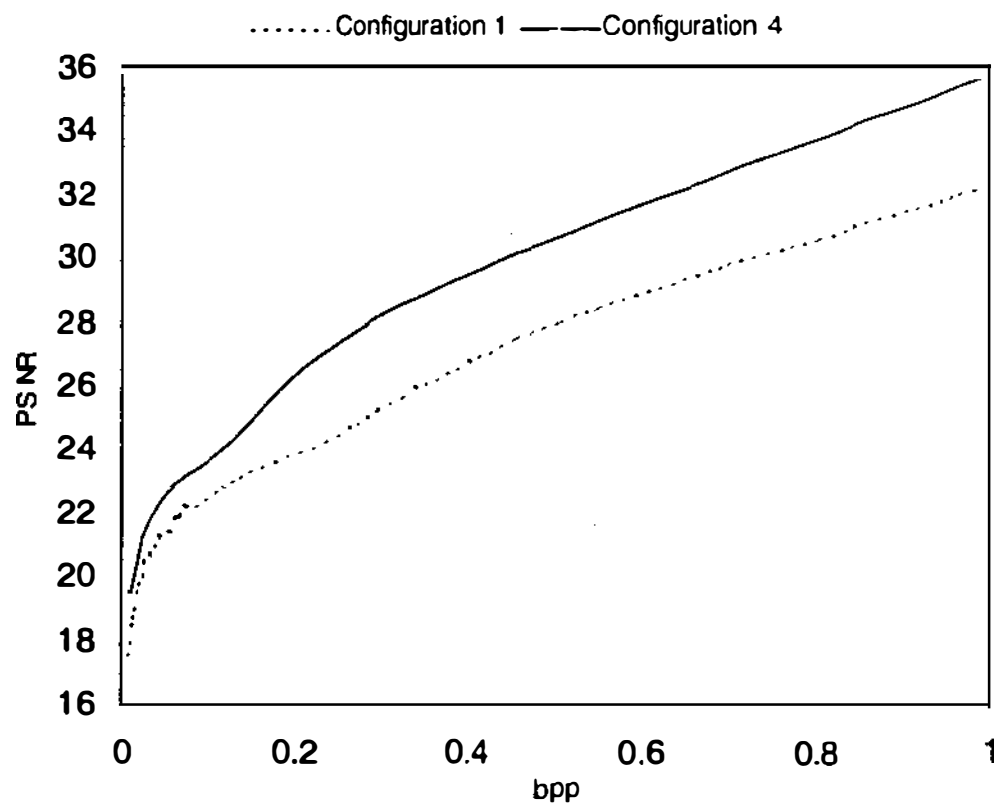
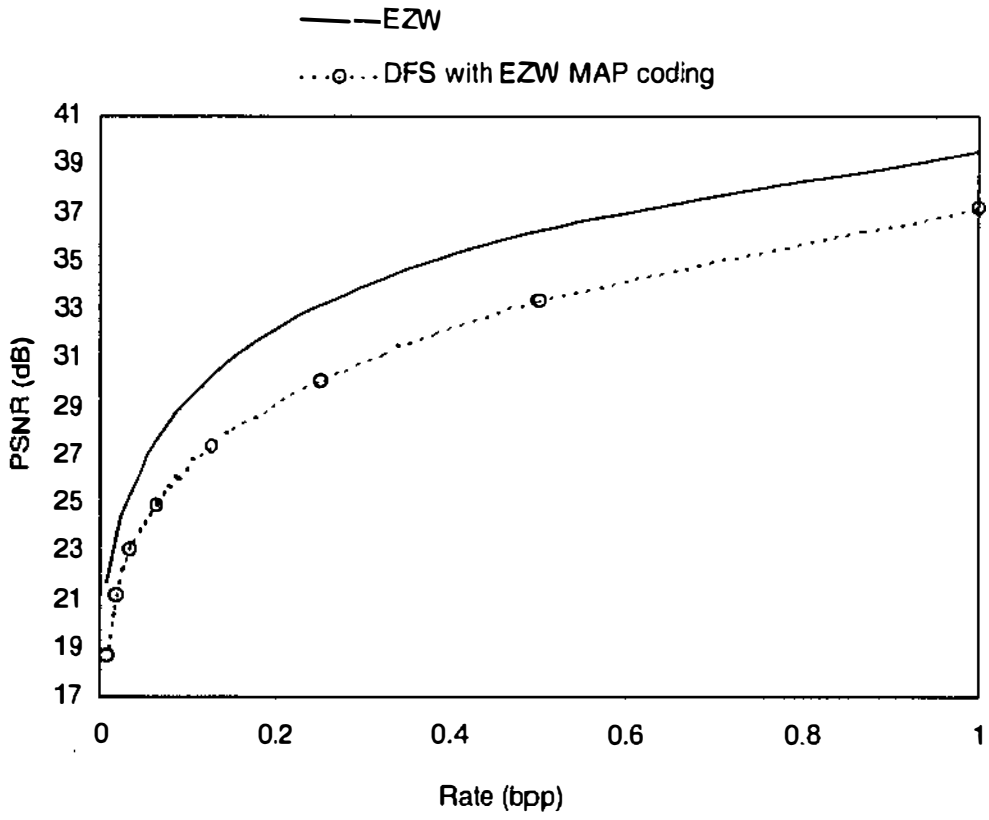


Figure 2.20 Simulation results for DFS Configurations for Barbara image

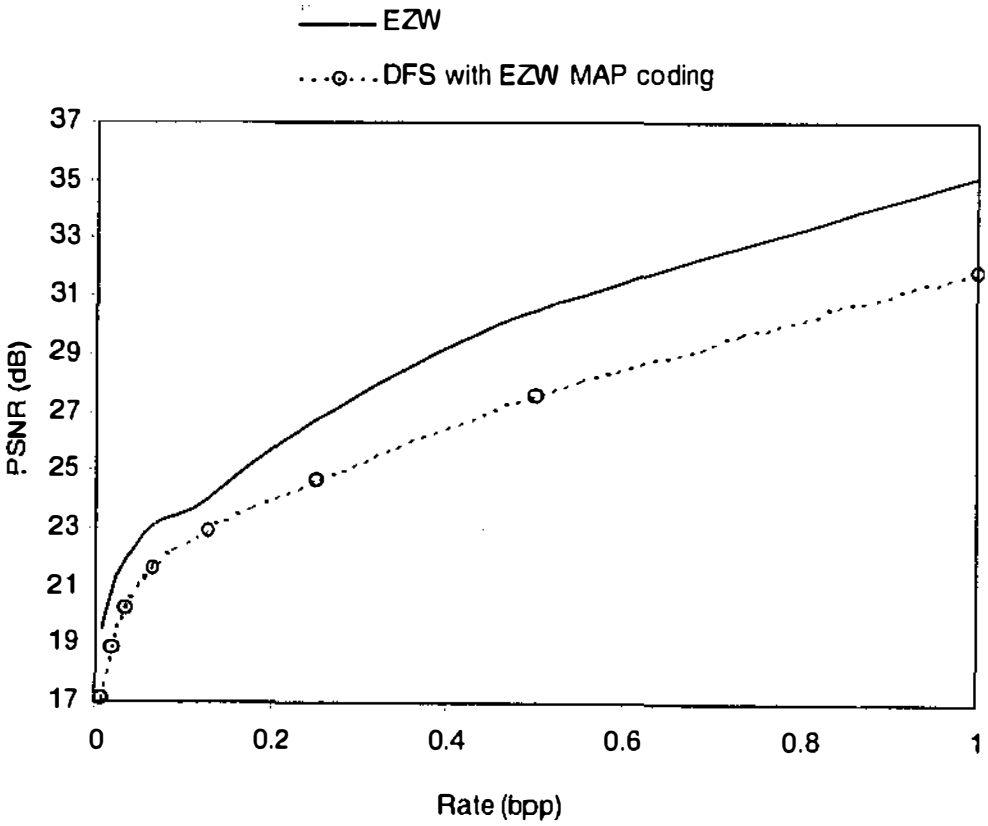
2.4.3 Comparison of MAP Coding Schemes

We next performed simulations to evaluate the performance of the different MAP coding schemes. We used the DFS in these simulations. We used the MAP coding schemes for the EZW, Improved EZW and SPIHT algorithms. The simulations were performed without arithmetic coding and without SAQ reordering. Figure 2.21, Figure 2.22 and Figure 2.23 show the results which were obtained for the EZW, Improved EZW and SPIHT MAP coding schemes respectively. The performances for the original EZW, Improved EZW and SPIHT algorithms are shown for comparison. The performance of the Improved EZW used the results of the Significance Checking in Wavelet Trees (SCIWT) algorithm reported in [19]. Six scales of subband decomposition using the 9/7 biorthogonal filter [49] were used and the integer part of the wavelet coefficients were converted to their sign-magnitude representation.

The EZW coefficient tree structure was used for the DFS EZW and the DFS Improved EZW coding schemes. The SPIHT coefficient tree structure was used for the DFS SPIHT coding scheme. The SDF symbol was not explicitly sent for MSIG nodes for the DFS EZW coding scheme. The SDF symbol was explicitly sent for MSIG nodes for the DFS Improved EZW and DFS SPIHT coding schemes. The AOMSIG symbol was used in all cases. For the various bit rates, the MSIG symbol was treated as the last received SAQ symbol after the end of transmission.

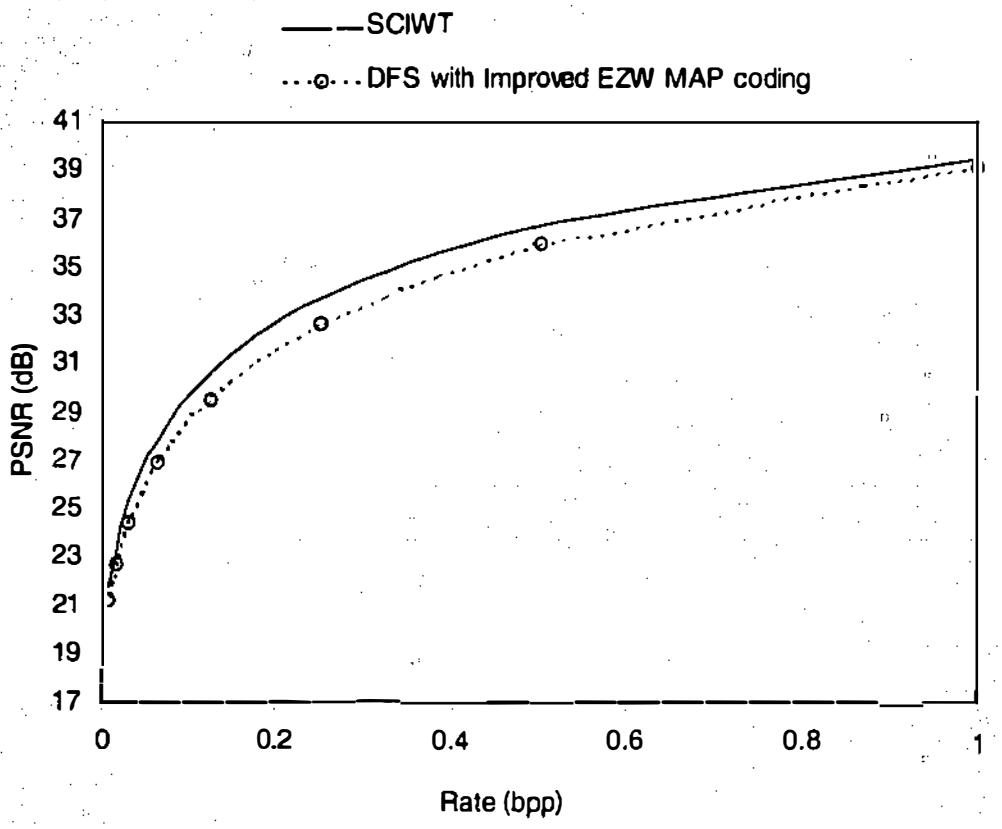


(a) Lena image

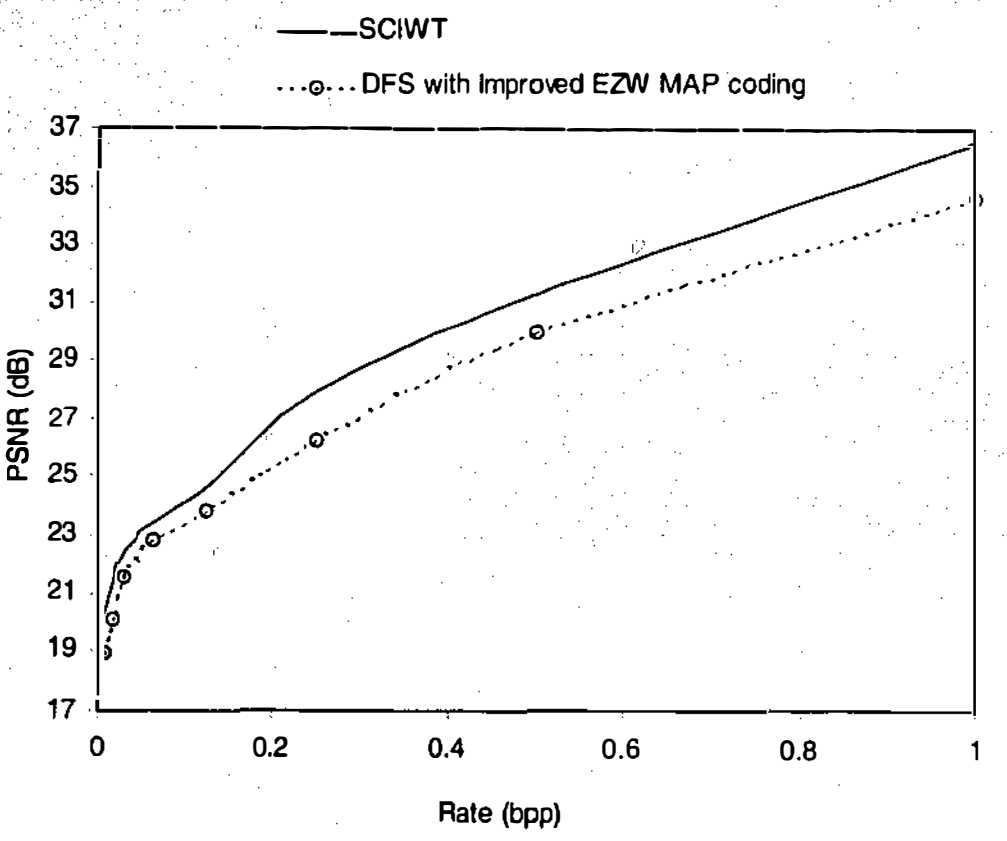


(b) Barbara image

Figure 2.21 Simulation results for DFS traversal with EZW MAP coding

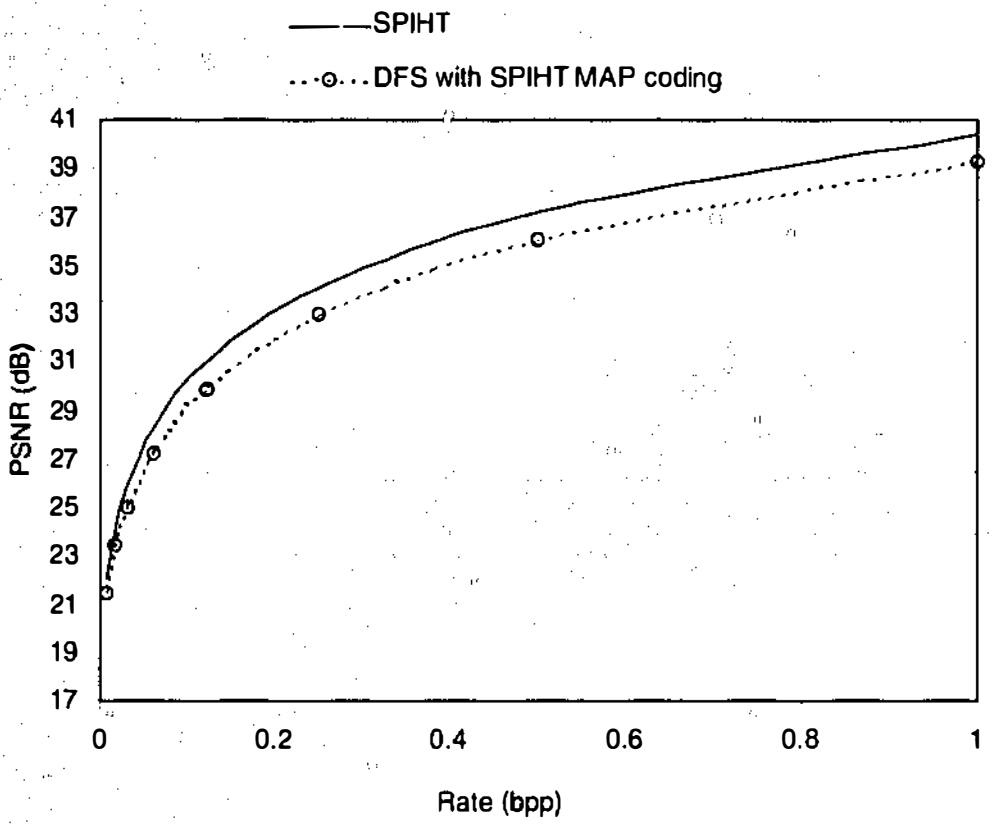


(a) Lena image

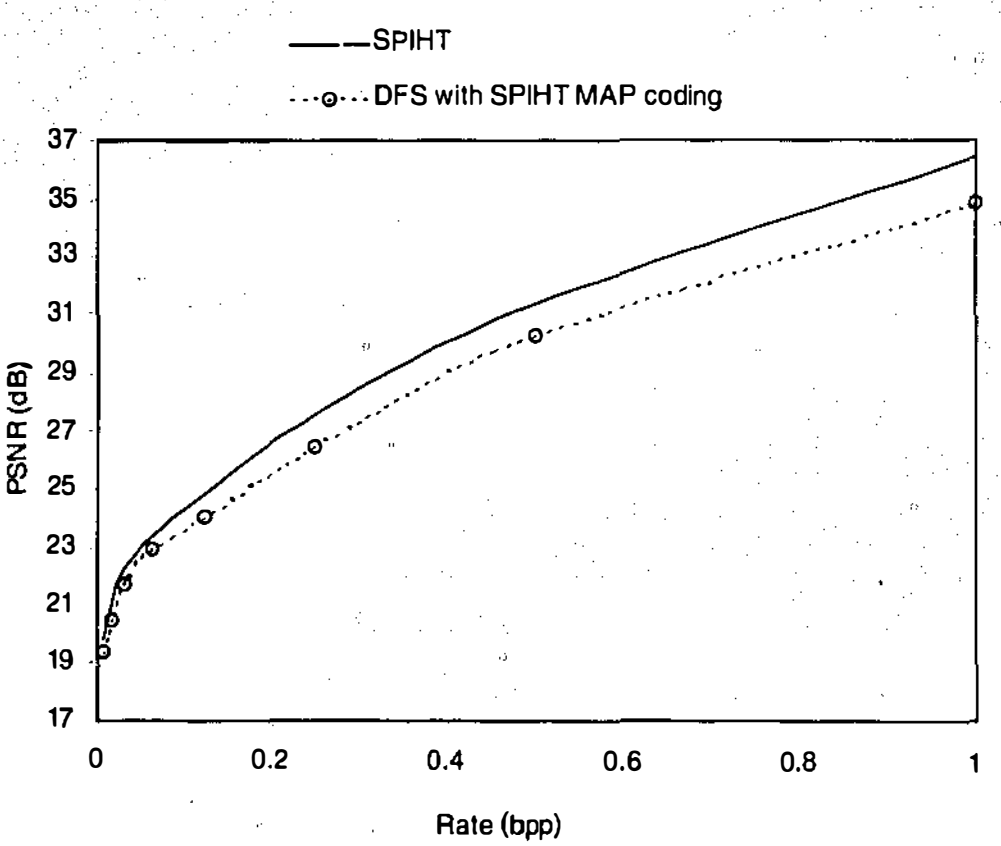


(b) Barbara image

Figure 2.22 Simulation results for DFS traversal with Improved EZW MAP coding



(a) Lena image



(b) Barbara image

Figure 2.23 Simulation results for DFS traversal with SPIHT MAP coding

The results show that even without using arithmetic coding, the performance of the DFS Improved EZW and DFS SPIHT algorithms are only slightly lower than their respective EW algorithms. There is a significant decrease in performance for the DFS EZW algorithm without arithmetic coding. The average decreases in performance are 2.4 dB, 1.0 dB and 0.9 dB for the DFS EZW, DFS Improved EZW and DFS SPIHT algorithms respectively. Other than the DFS EZW algorithm, the performance of the DFS EW algorithms are only about one dB less than their full algorithms. The performance of the DFS SPIHT algorithm is almost comparable to the performance of the full EZW algorithm.

Finally, we performed simulations to investigate the DFS SPIHT algorithm further. For these simulations, we replaced the SPIHT tree structure with the EZW tree structure and did not use the AOMSIG symbol. The AOMSIG symbol was found to increase performance only for bit rates above 1 bpp. We used the test images as shown in Figure 2.24 and averaged the PSNR for the five images. Figure 2.25 shows the results which were obtained. The performance of the SPIHT algorithm with and without arithmetic coding (AC) are shown for comparison. The average decreases in performance for the DFS SPIHT algorithm with the SPIHT algorithm without arithmetic coding and with arithmetic coding are 0.6 dB and 1.0 dB respectively.

Figure 2.24 Test images

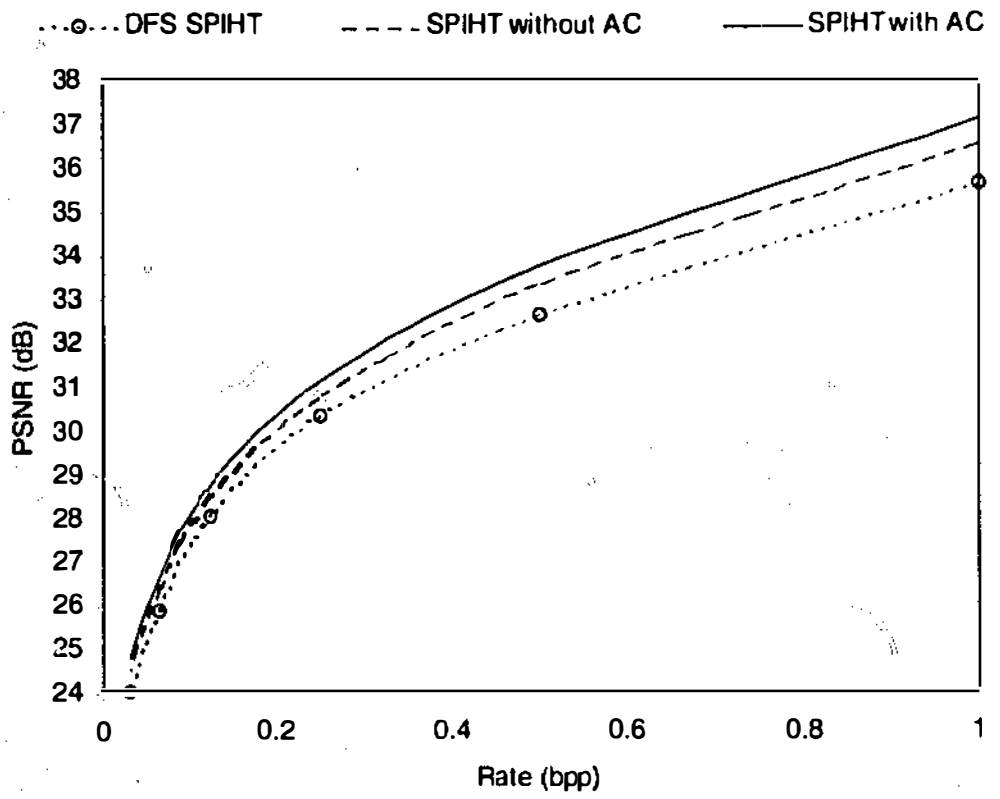


Figure 2.25 Simulation results for DFS SPIHT MAP coding

2.5 Conclusions

In this chapter, we have outlined the variations and options for EW algorithms and discussed the variations which are suitable for hardware implementation. EW algorithms differ in the structure of the coefficient tree used, the strategy to search the coefficient tree for significant coefficients, the scheme to code the coefficient tree and the generation and transmission of the symbol stream so that symbols which are more significant are transmitted ahead of less significant symbols. For the coding of the coefficient tree, we have introduced a set of MAP symbols which is applicable for a number of EW algorithms.

We have presented a variation of EW algorithms for hardware implementation based on DFS processing. The DFS EW algorithms have lower storage requirements than

previous approaches, simpler computations and allows for efficient addressing of the nodes in the coefficient tree. The partitioning of the coefficient tree into subtrees, the depth-first search propagation of the significance symbols, MSIG quantization and bit stream processing gives these advantages for hardware implementation. Simulations show that even without using arithmetic coding, the performance of the DFS Improved EZW and DFS SPIHT variants are only slightly lower than their complete algorithms. The performance of the DFS SPIHT variant is almost comparable to the performance of the complete EZW algorithm. Comparisons with the WVQ algorithm [33] show that the performance of the DFS SPIHT algorithm is higher than the WVQ algorithm at high bit rates and is comparable to the WVQ algorithm at lower bit rates. Table 2.7 shows a summary of the DFS SPIHT and WVQ algorithms. The DFS EW variants offer algorithms which are suitable for implementation in hardware without significant decreases in the performance of the coding system.

Table 2.7 Comparison of DFS SPIHT and WVQ algorithms

	DFS SPIHT	WVQ
Tree searching scheme	iterative	noniterative
Quantization method	scalar	vector

As a final note, we would like to say that the DFS approach has been developed with some foreknowledge of hardware architectures which would be suitable. In particular, we have in mind the bit stream architectures which we will discuss in the next chapter. The feature of these architectures is that the coefficient bits are not stored in the processor and are processed as they flow through. This results in fast architectures which have low complexity for hardware implementation. The DFS EW algorithms are distinctive from previous approaches for hardware implementation because of their corresponding bit stream architectures which are readily available for implementation.

Chapter 3

Architectures for the DFS Embedded Wavelet Algorithms

3.1 Introduction

Embedded wavelet (EW) algorithms vary from one another in the structure of the coefficient tree, the strategy to search the coefficient tree for significant coefficients, the scheme to code the coefficient tree and the generation and transmission of the symbol stream so that symbols which are more significant are transmitted ahead of less significant symbols. Amongst the variations, the tree searching (TS) strategy and the tree coding scheme can be considered to be the two variations which give each EW algorithm its particular characteristic. One variation for TS is the depth-first search (DFS). The DFS simplifies the hardware implementation of EW algorithms. The DFS:

- Performs a natural partitioning of the coefficient tree into subtrees which can be processed independently and minimizes the storage requirements by processing one subtree at a time;
- Provides an efficient scheme to determine ancestor-descendant relationships in the subtrees by propagating the significance symbols to the ancestor node as soon as the necessary descendant nodes have been searched.

The processing requirements can be further simplified by using the sign-magnitude binary representation of the coefficients in the coding process. The use of the binary representation in the coding process translates into using threshold values which are powers of two for significance map (MAP) coding. In addition, successive-approximation quantization (SAQ) of significant coefficients is simplified to magnitude

(MAG) coding where the SAQ symbols can be obtained directly from the magnitude representation of the coefficients.

Previous approaches to implement EW algorithms in hardware have been mainly by making use of memory banks to store the wavelet coefficients where all the coefficients can be accessed to establish the ancestor-descendant relationships [34], [35], [36], [37]. The disadvantage of the memory bank approach is that it requires the coefficients to be stored in the TS processor prior to processing. In contrast to the all memory bank approach, we propose a new architecture to implement the tree searching. In this approach, the wavelet coefficients of an image are transferred from the two-dimensional discrete wavelet transform (DWT) processor to the TS processor in a bit stream which can be processed immediately. For clarity of discussion, we will look at the bit stream (BS) architecture in relation to the implementation of the DFS EZW algorithm. The approach can be extended for the implementation of other DFS algorithms. A second issue we will discuss is the flexibility of the DFS BS architectures to implement variations in EW algorithms. In particular, we will look at how the DFS BS architectures can be modified to implement tree searching strategies which are similar in principle to the set partitioning scheme used in the SPIHT algorithm. For the discussion, we will make use of a parallel DFS BS EZW architecture.

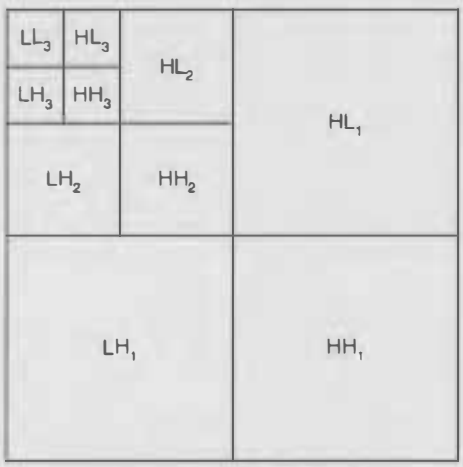
This chapter is organized as follows. In Section 3.2, we discuss the DFS bit stream architecture for the implementation of EW algorithms. The DFS BS architecture results in simple and fast implementations of the algorithms. We discuss the DFS BS parallel architecture in Section 3.3. The parallel approach results in an even faster architecture at the cost of an increase in hardware complexity. In addition, the parallel architecture

provides options to output for different tree searching formats to achieve higher coding performances. Conclusions are presented in Section 3.4.

3.2 Bit Stream Architecture

3.2.1 Depth-First Search Bit Stream Input

Figure 3.1(a) shows the subband arrangement for a three scale DWT decomposition and Figure 3.1(b) shows an example of the wavelet coefficients used in EZW [16].



(a) Subband arrangement

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

(b) Wavelet coefficients

Figure 3.1 Subband arrangement and example of coefficients

Figure 3.2 shows the tree representation of the coefficients and Figure 3.3 shows the order of the wavelet subbands which will be searched by the DFS. The DFS partitions the coefficient tree into a number of subtrees each of which has one root node. The number of subtrees is equal to the number of coefficients in the lowest frequency subband. Each subtree can be further partitioned into its four subband orientations. The lists of coefficients in the four orientations are concatenated in the order LL, HL, LH and HH to a single list representing the entire subtree.

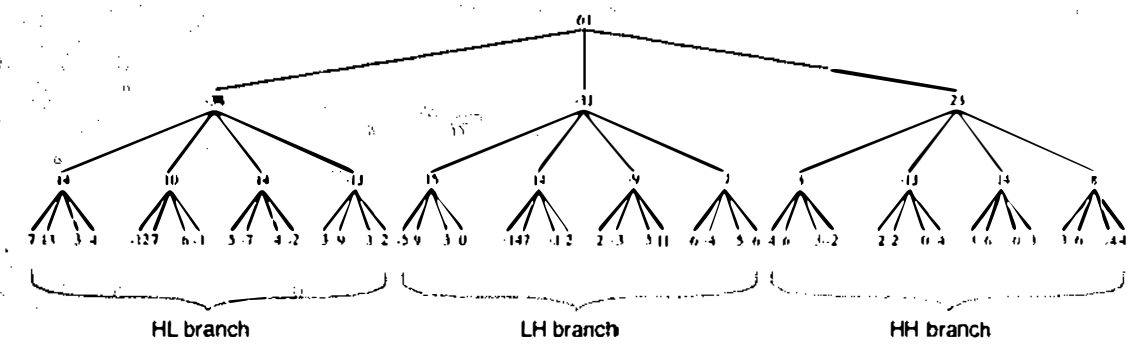


Figure 3.2 Coefficient tree example from EZW

Level	Subband orientation			
	LL	HL	LH	HH
0	63			
1		-34	-31	23
2		49	15	3
3		7	-5	4
3		13	9	6
3		3	3	3
3		4	0	-2
2		10	14	-12
3		-12	-1	-2
3		7	47	2
3		6	-3	0
3		-1	2	4
2		14	-9	-14
3		5	2	3
3		-7	-3	6
3		4	5	0
3		-2	11	3
2		-13	-7	8
3		3	6	3
3		9	-4	6
3		3	5	-4
3		2	6	4

Figure 3.3 DFS representation showing subband orientations

Figure 3.4 shows the binary representation of the coefficient tree in Figure 3.2 in the DFS format and the arrangement of the bit stream into the DFS BS EZW processor. The coefficients are represented using the sign-magnitude representation. The coefficient magnitudes are represented using seven bits. The MSB is represented as b_6

and the LSB is represented as b_0. The sign bits are represented as b_7. All the sign bits for the coefficients are shifted out first followed by the MSBs and ending with the LSBs. The bits are input beginning from the leaf coefficients and ending at the root coefficient of the tree.

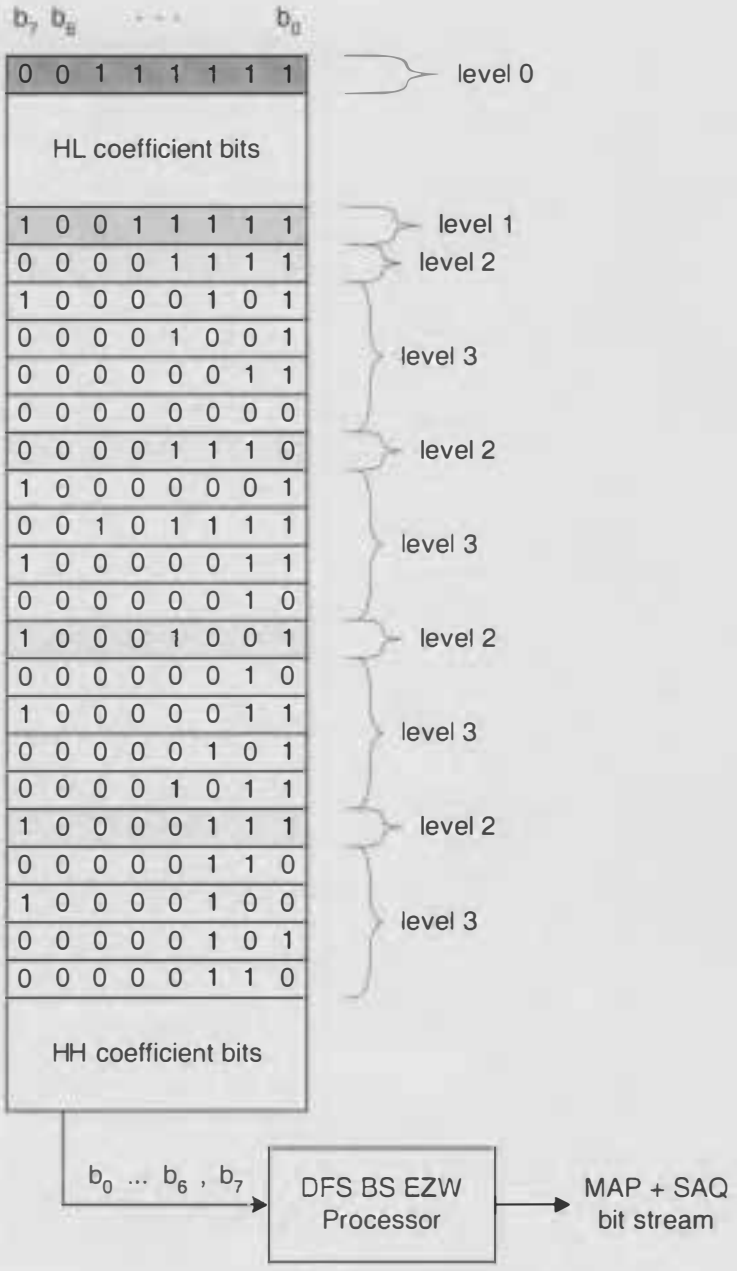


Figure 3.4 DFS input bit stream

3.2.2 Encoder Architecture

Figure 3.5 shows the DFS BS EZW encoder architecture showing the flow and storage of information for a coefficient tree with N subtrees. Each subtree has NC coefficient nodes.

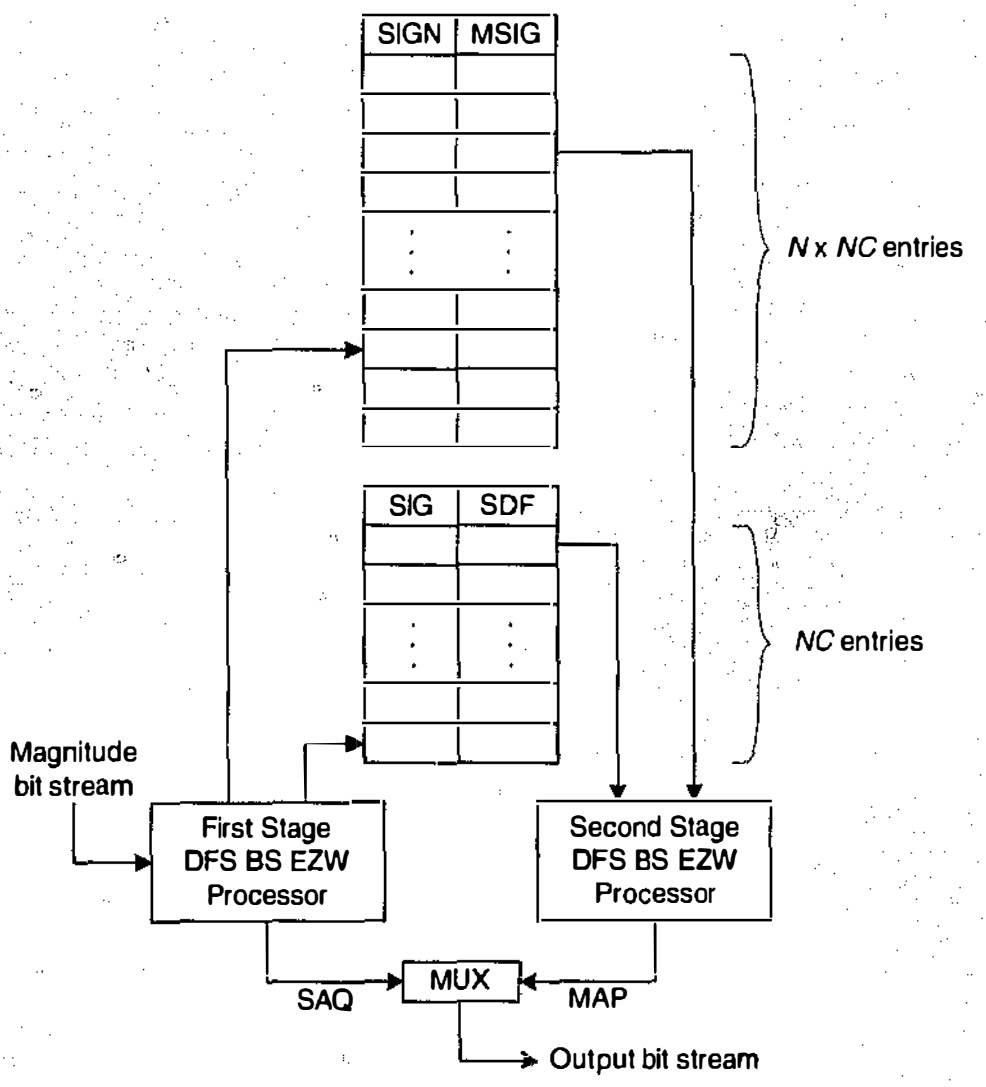


Figure 3.5 DFS BS EZW encoder architecture

The sign bits are input first into the processor and stored. An option would be to not store the sign bits and to transmit them to the decoder immediately. This has the drawback of sending sign bits for coefficients which would not be significant for the target bit rate and lead to a reduction in coding performance. For a high target bit rate

when many coefficients would be significant, this may be a viable option. But for a lower target bit rate, this would cause a significant decrease in coding performance. In our architecture, we use a general scheme where the sign bits are stored in the processor.

Whereas the sign bits have to be stored in the processor, the magnitude bits can be discarded once they are processed. Two stages are required for every pass of the magnitude bits at the same bit plane. The two stages correspond to scanning the coefficient tree twice for each threshold value. The First Stage scans the coefficient tree from the leaf nodes up to the root nodes and the Second Stage scans the coefficient tree from the root nodes down to the leaf nodes. The First Stage involves processing the magnitude bits as they arrive at the processor. The corresponding coefficient is determined to be significant if the bit arriving at the processor is 1; otherwise it is not significant. Using the MAP coding scheme, this is represented by the SIG symbol. A coefficient is found significant only once and this is represented by the MSIG symbol. A MSIG coefficient is treated as having a zero magnitude by the other coefficients. At the same time, the SDF symbol for each coefficient in the current pass is set if the coefficient has at least a significant descendant.

Another function performed in the First Stage is to generate and to output the SAQ bit for each coefficient which has been marked significant in the previous passes. In the bit stream architecture, the SAQ bit is just the bit itself for the MSIG coefficient. The outcome after the First Stage DFS BS EZW processor is that the SIG and SDF MAP bits have been generated for non MSIG coefficients and the SAQ bits have been generated for MSIG coefficients.

The Second Stage involves scanning the information produced in the First Stage of processing to output the MAP symbols. The MAP symbol assignment for the EZW coefficient nodes is shown in Table 3.1. The symbol POS has the binary representation of 10, NEG of 11, IZ of 01 and ZTR of 00 where the first bit is the SIG bit.

Table 3.1 Assignment of MAP symbols for EZW coefficient nodes

Symbol assigned	SIGN	SIG	SDF
POS	0	1	X
NEG	1	1	X
IZ	X	0	1
ZTR if NZF = 1	X	0	0

The ZTR symbol is only output if the coefficient ancestor is not assigned a ZTR symbol. This condition is indicated by a flag, Not Zerotree Root Flag (NZF) which is set to 1 if the ancestor of the coefficient is not a ZTR symbol. Coefficients marked significant will not have MAP symbols in the subsequent passes. Each entry in the storage in the processor for each coefficient position has the following fields:

| SIGN | SIG | SDF | MSIG | NZF |

Due to the depth-first nature, NZF does not need to be kept for a coefficient throughout the search but only a single NZF field is required for each level in the coefficient tree. Similarly, the SIG and SDF fields do not need to be kept for all $N \times NC$ coefficients but only for the NC coefficients of a single subtree. For a coefficient tree containing N subtrees where each subtree has NC coefficient nodes, the total storage required is $2 \times N \times NC$ bits for storing the SIGN and MSIG bits for all the coefficients and $2 \times NC$ bits for storing the SIG and SDF bits for the subtree being processed. Further reduction in storage may be possible by making use of the storage in the DWT

processor for SIGN, SIG and MSIG for the subtrees which are not being processed currently.

3.2.2.1 First Stage Processor

Central to the First Stage is the generation and storage of the SIG and SDF MAP bits for the coefficients. If at each clock cycle i, one magnitude bit MAG is input into the processor then the MAP bits can be determined by the following Equations:

$$\text{SIG}(i) = \text{MAG}(i) \text{ AND NOT}(\text{MSIG}(i)) \tag{3.1}$$

$$\text{SDF}(i) = \text{SIG}(i\text{-}1) \text{ OR SIG}(i\text{-}d\text{-}1) \text{ OR SIG}(i\text{-}2d\text{-}1) \text{ OR SIG } (i\text{-}3d\text{-}1) \text{ OR SDF}(i\text{-}1) \text{ OR SDF}(i\text{-}d\text{-}1) \text{ OR SDF}(i\text{-}2d\text{-}1) \text{ OR SDF } (i\text{-}3d\text{-}1) \tag{3.2}$$

where d is the number of bits which have to be delayed from a child coefficient to its next sibling coefficient and is given by:

$$d = \frac{4^{l_{max} - l_{cur}} - 1}{3} \tag{3.3}$$

The labels l_{max} and l_{cur} denotes the maximum level and the current level in the coefficient tree respectively. The values of d are 21, 5 and 1 for levels 0, 1 and 2 respectively. Equation (3.1) shows that the SIG bit is set to the input magnitude bit for nodes which are not MSIG. For MSIG nodes, the SIG bit is set to 0. Equation (3.2) shows that the SDF of a coefficient is determined by the SIG and SDF bits of its children. For example, the SDF for a coefficient node at level 2 is determined by the SDF and SIG bits of its children nodes at level 3. Figure 3.6 shows the architecture for the First Stage Processor. The LEVEL SELECT circuit selects the current level of the incoming magnitude bit. For each subtree, the circuit generates the DFS sequence

11(3), 11(3), 11(3), 11(3), 10(2), 11(3), 11(3), 11(3), 11(3), 10(2), 11(3), 11(3), 11(3), 11(3), 10(2), 11(3), 11(3), 11(3), 11(3), 10(2), 01(1), ..., 01(1) ,00(0) corresponding to the current levels shown inside parenthesis. The circuit can be implemented using a two-bit counter for each level. The SDF values for leaf nodes corresponding to the level sequence 11 will always be 0.

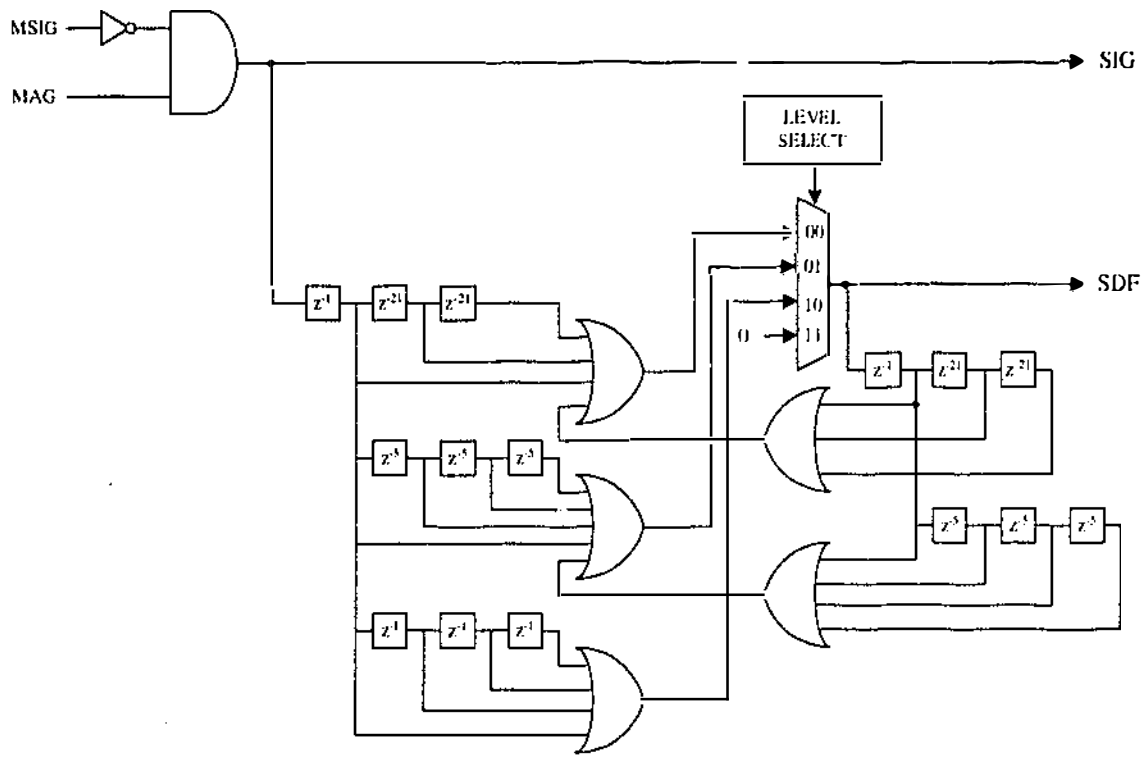


Figure 3.6 First Stage Processor architecture

Figure 3.6 shows that several one-bit storage and OR gates are required in the architecture. In practice, in order to assign a value to SDF, only a single bit at each level is needed to keep the temporary value for the level above. Let these bits be TSDF(l). The function of these bits is to keep the temporary SDF for each level. Initially, these bits are set to zeros. It is assigned to SDF(l-1) and reset after the coefficients in a subtree at level l are processed during the search. Figure 3.7 shows an example of a SIG bit plane and the two TSDF bits required for processing.

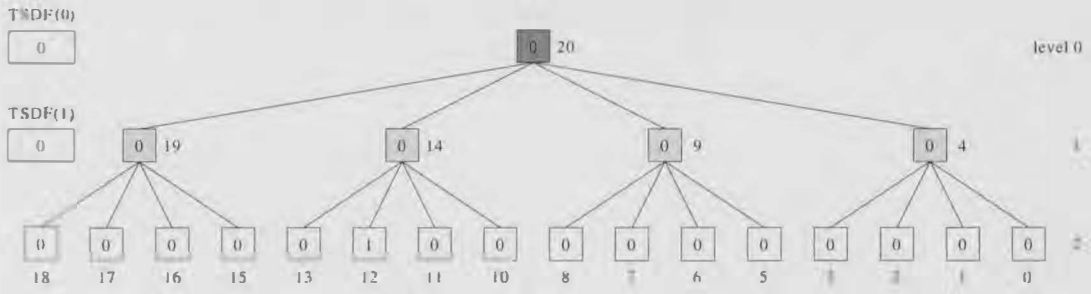


Figure 3.7 Example of a SIG bit plane

Table 3.2 Assignment of TSDF bits

i	MAG	LEVEL SELECT	TSDF(0)	TSDF(1)	SIG	SDF
0	0	10	0	0	0	0
1	0	10	0	0	0	0
2	0	10	0	0	0	0
3	0	10	0	0	0	0
4	0	01	0	0	0	0
5	0	10	0	0	0	0
6	0	10	0	0	0	0
7	0	10	0	0	0	0
8	0	10	0	0	0	0
9	0	01	0	0	0	0
10	0	10	0	0	0	0
11	0	10	0	0	0	0
12	1	10	0	1	1	0
13	0	10	0	1	0	0
14	0	01	1	0	0	1
15	0	10	1	0	0	0
16	0	10	1	0	0	0
17	0	10	1	0	0	0
18	0	10	1	0	0	0
19	0	01	1	0	0	0
20	0	00	0	0	0	1

Table 3.2 shows the assignment of the TSDF bits. Figure 3.8 shows the flowchart to accomplish the First Stage Processing. Note that TSDF(0) is not immediately assigned the value of 1 for the incoming magnitude bit at the 13th clock cycle (i = 12). It is only assigned the value of 1 at the 15th clock cycle (i = 14) when the level above is being processed. In this way, the SDF bits are propagated from the leaves to the root node. If

any node below the root has a SIG value of 1, TSDF(0) will be set to 1; else TSDF(0) will remain 0. TSDF(0) is reset after the root node has been processed whereas TSDF(1) is reset for processing a new branch.

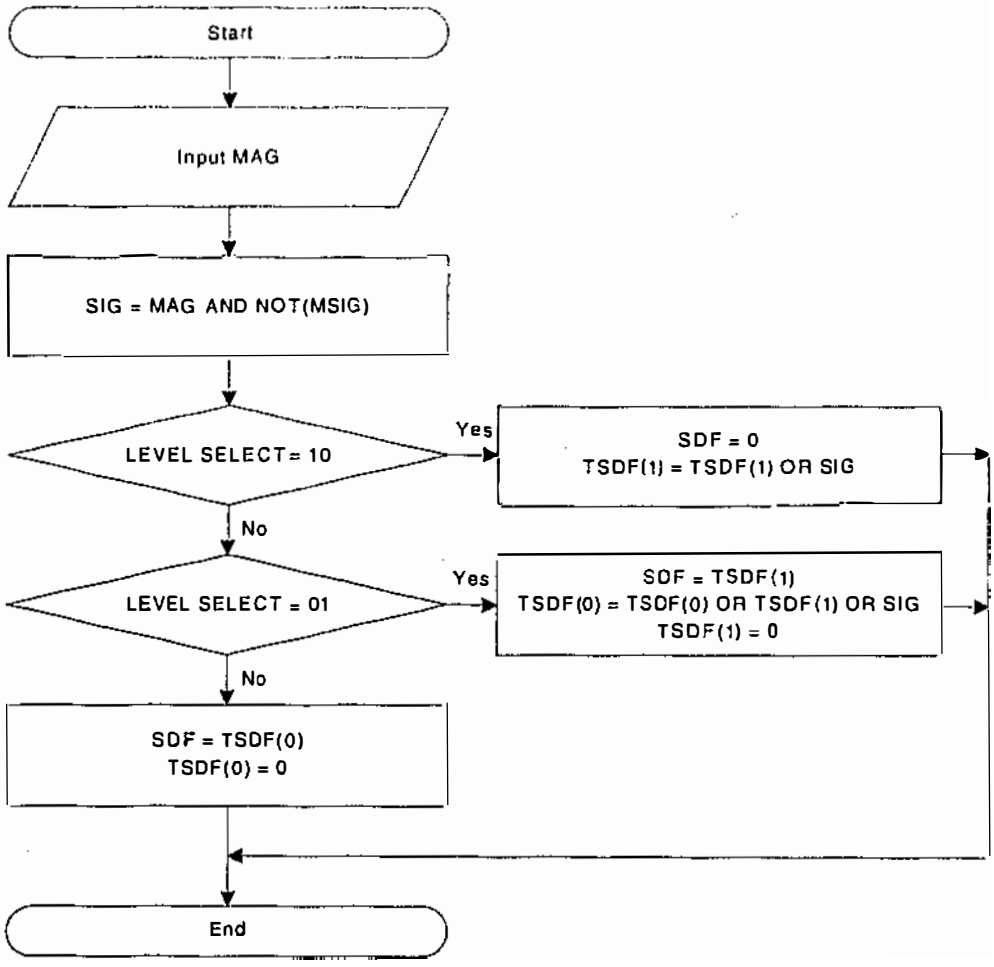


Figure 3.8 Flowchart for First Stage Processing

3.2.2.2 Second Stage Processor

The Second Stage is to scan from the top of the storage in the processor to the bottom. Two flags TNZF(*l*) (Temporary NZF) need to be kept at each level to indicate if a coefficient parent is not a ZTR node. Only one bit is needed at each level since it is a DFS. Figure 3.9 shows an example of a ZTR bit plane and the two TNZF bits required for processing. A value of 1 indicates that the node is a ZTR node. Table 3.3

shows the assignment of the TNZF bits. Figure 3.10 shows the flowchart to accomplish the Second Stage Processing.

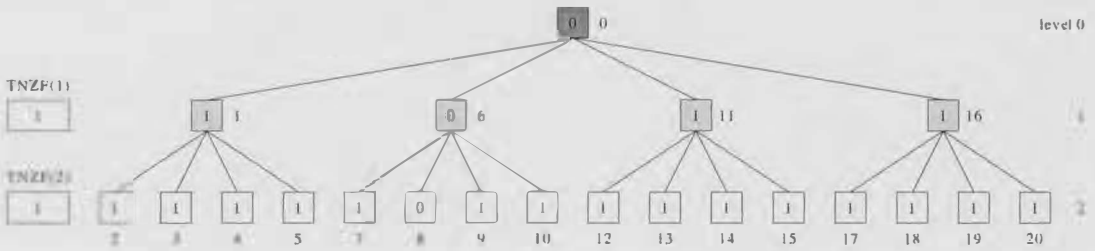


Figure 3.9 Example of a ZTR bit plane

Table 3.3 Assignment of TNZF bits

i	ZTR	LEVEL SELECT	TNZF(1)	TNZF(2)	NZF
0	0	00	1	1	1
1	1	01	1	0	1
2	1	10	1	0	0
3	1	10	1	0	0
4	1	10	1	0	0
5	1	10	1	0	0
6	0	01	1	1	1
7	1	10	1	1	1
8	0	10	1	1	1
9	1	10	1	1	1
10	1	10	1	1	1
11	1	01	1	0	1
12	1	10	1	0	0
13	1	10	1	0	0
14	1	10	1	0	0
15	1	10	1	0	0
16	1	01	1	0	1
17	1	10	1	0	0
18	1	10	1	0	0
19	1	10	1	0	0
20	1	10	1	0	0

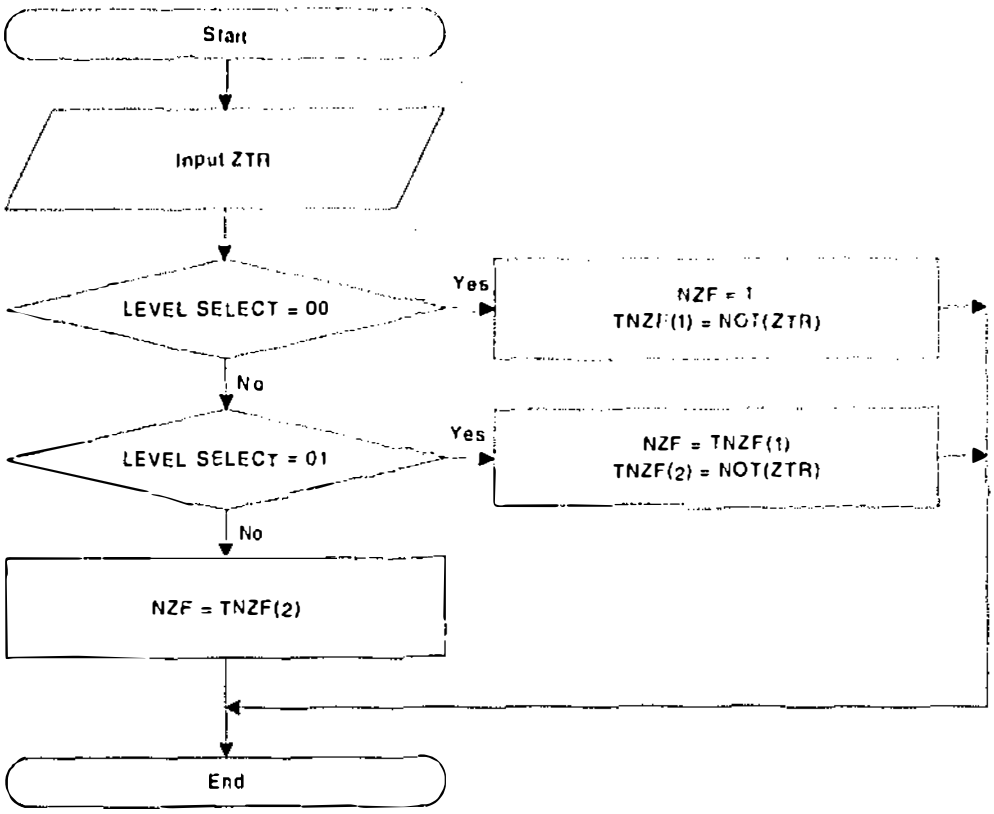
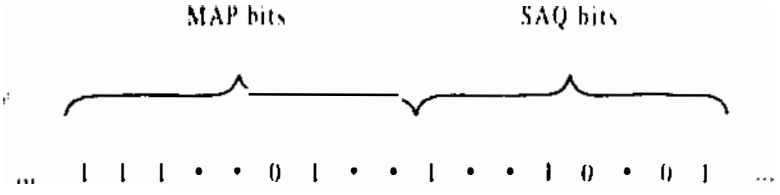


Figure 3.10 Flowchart for Second Stage Processing

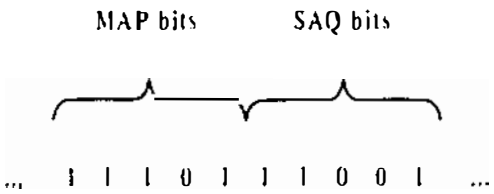
3.2.3 Decoder Architecture

In this section, we discuss the decoder architecture for the bit stream approach. In particular, we will discuss a consideration because of modifications which are performed to the bit stream due to channel coding. In the DFS BS EZW architecture shown in Figure 3.5, the SAQ bits are output by the First Stage Processor and the MAP bits are output by the Second Stage Processor. The MAP and SAQ bits form the output bit stream. The SAQ bits are only output for coefficients which are MSIG whereas the MAP bits are only output for coefficients which are not MSIG. Also, ZTR MAP bits are only output if the NZF flag is 1. This means that there are clock cycles for which no MAP or SAQ bits are output. Figure 3.11(a) shows an example of the output bit stream from the encoder where • represents the no-output case. During channel coding, the

channel will compact together the bits for transmission to be sent to the decoder. Figure 3.11(b) shows the input bit stream to the decoder.



(a) Uncompacted output bit stream from the encoder



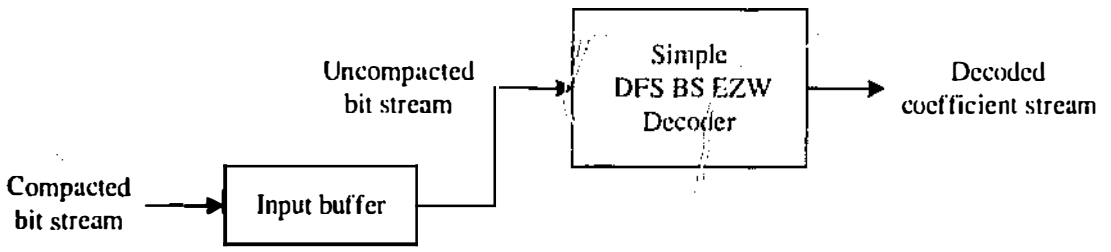
(b) Compacted input bit stream into the decoder

Figure 3.11 Bit streams before and after channel coding

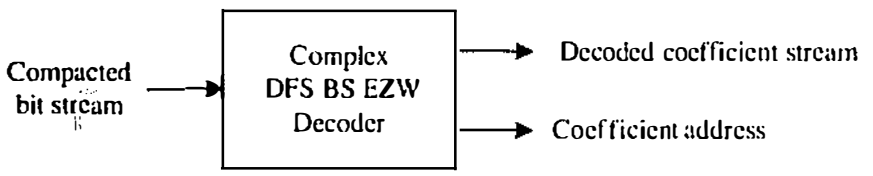
The architecture to decode the output bit stream in Figure 3.11(a) back into the coefficient bit stream is simpler than the encoder architecture because for each bit plane only one stage of processing is required. The processing corresponds to scanning the coefficient tree from the root nodes down to the leaf nodes. The complexity arises because of the compaction to the bit stream due to channel coding. In the bit stream architecture, bits which arrive have to be processed immediately. If the bits are not processed immediately, they have to be stored until they are ready to be processed. Due to the compaction performed on the bit stream by the channel coder, the bits would arrive at the decoder earlier than the decoder expects.

We propose two approaches to handle the discrepancies in timing between the transmission channel and the decoder. The first approach is to buffer the incoming bits to regain back the uncompacted output bit stream as in Figure 3.11(a). This approach

leads to a simple decoder architecture at the cost of an additional input buffer which is required. The second approach is to utilize a more complex decoding scheme where each bit which is output from the decoder is accompanied by an address which informs the wavelet processor of the position of the coefficient in the tree. This approach would require a decoder with more complexity but would not require the input buffer. In addition, this approach leads to a faster decoder architecture. Figure 3.12(a) and Figure 3.12(b) show the two approaches for the DFS BS EZW decoder architecture.



(a) Simple decoder architecture



(b) Complex decoder architecture

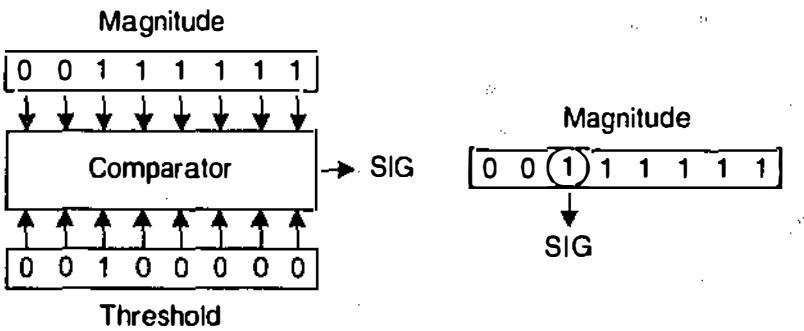
Figure 3.12 Approaches for the DFS BS EZW decoder architecture

3.2.4 Advantages of DFS BS Architecture

The bit stream approach of processing the bits as they arrive at the processor leads to simple architectures with minimal memory requirements. An additional advantage is easier interfacing to the DWT processor and the transmission channel.

3.2.4.1 Simple Comparison Operations

The determination of the SIG symbol can be obtained directly from the magnitude representation of the coefficient without having to perform any comparison operations. Figure 3.13 shows the determination of the SIG symbol for the coefficient 63 using a threshold of 32. Figure 3.13(a) shows the comparate approach used to determine coefficient significance used in the architectures in [34], [37] and Figure 3.13(b) shows the bit stream approach. Using the bit stream approach, the SIG symbol is simply given by the current magnitude bit. The difference between the two approaches is that the bit stream approach uses threshold values which are powers of two.



(a) Comparator approach (b) Bit stream approach

Figure 3.13 Calculation of SIG symbol using bit stream approach

3.2.4.2 Simple Arithmetic Operations

One requirement in decoding is the calculation of the approximated coefficient value after the end of transmission. Figure 3.14 shows the calculation of the coefficient 63 after three threshold values. When the decoder receives the POS symbol, it decodes the sign bit and the MSB and sets the MSIG bit to 1, but does not insert it in position yet. For the thresholds of 16 and 8, the decoder receives the two SAQ '1' bits. If transmission ends here, the decoder inserts then the MSIG bit into the next available bit

position before decoding the coefficient value. Zeros are inserted into the remaining bit positions. The decoded value is 011100 which is +60 in sign-magnitude representation. The decoded value can be obtained without having to perform any arithmetic operations.

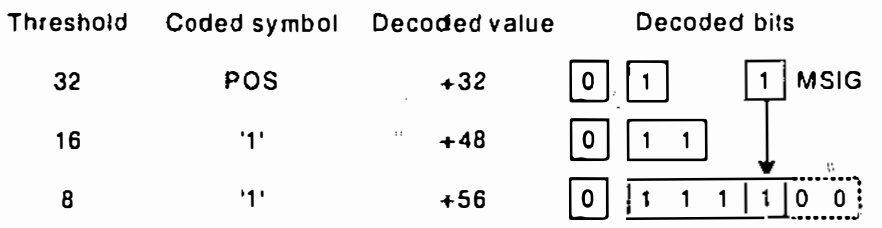


Figure 3.14 Calculation of approximated coefficient value

3.2.4.3 Minimal Storage Architecture

The bit stream architecture operates on individual bits at a time. Each bit in the coefficient binary representation is treated as a separate piece of information. The MSBs are processed and discarded before the LSBs are even input into the architecture. This feature of processing individual bits differentiates the bit stream approach from other architectures which operate on the entire coefficient binary representation [34], [37]. The bit stream approach also results in an architecture with minimal storage requirements. The same storage for processing the MSBs can be reused for processing the LSBs.

3.2.4.4 Just-In-Time Processing

The bit stream architecture gives the flexibility of processing as many or as few bits to suit a transmission bit rate. For example, Figure 3.15 shows a DFS BS EZW processor with a channel rate control. When the target channel rate is met, the DFS BS EZW processor is informed and future incoming bits are not processed. The processor

doesn't process any further if the channel cannot accommodate the generated bits. In this way, no processing is wasted. For comparison, the other approach would be to process all incoming bits and discard the generated bits which cannot be accommodated by the transmission channel.

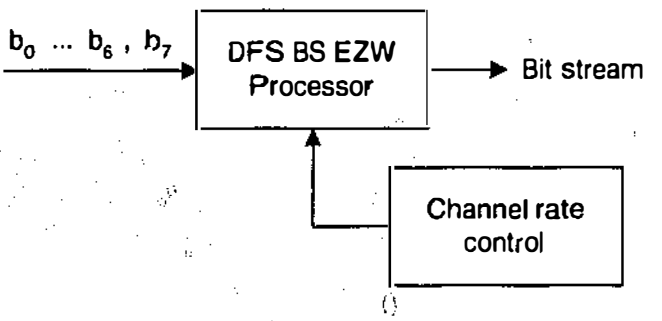


Figure 3.15 DFS BS EZW processor with channel rate control

3.2.4.5 Architecture Scalability

DWT processors have been designed to suit different image sizes. Another advantage of the bit stream architecture is that it can be used to handle images of varying dimensions provided that the same number of subband decompositions are used. A 16×16 image with three scales of subband decomposition would give four 8×8 subtrees whereas a 32×32 image with three scales of subband decomposition would give 16 8×8 subtrees as shown in Figure 3.16. As far as the architecture is concerned, the only difference between processing the two images is that the input bit stream of the larger image is four times longer than the input bit stream of the smaller image.

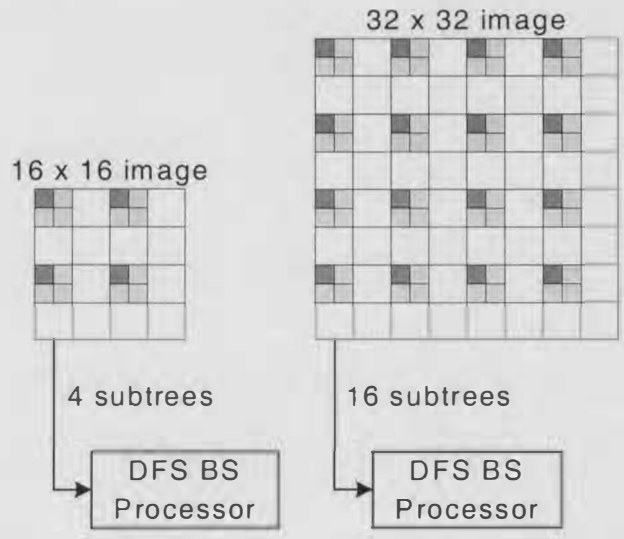


Figure 3.16 Scalability of bit stream architecture

3.3 Parallel Architecture

This section presents a parallel architecture for the implementation of the EZW algorithm based on the DFS BS architecture. Using the depth-first search of the wavelet coefficient tree, the wavelet coefficients in the coefficient tree are first partitioned into independent subtrees. In the case of full parallelism, each of the subtrees is processed by an independent processor. The output from each processor is then multiplexed back into a single output bit stream. While the output bit stream from each subtree processor is in the DFS format, the overall multiplexed output bit stream represents the search of the subtrees in parallel. The use of the DFS BS structure also makes it possible for partial parallelism where a subtree processor can process two or more subtrees in sequence. This provides flexibility for the design of the overall processor optimally to match the rate of the overall input bit stream. A subtree processor can be easily modified to perform any improved MAP coding scheme and the multiplexer for the output bit streams from the processors can be modified to produce the format of different TS strategies. In particular, we will look at how the multiplexer can be

modified to obtain the outputs from the parallel processors in such a way as to implement a tree searching strategy which is similar in principle to the set partitioning search strategy used in the SPIHT algorithm.

3.3.1 Parallel Implementation of the DFS BS EZW Algorithm

The maximum parallelism which can be achieved is when the number of processors is equal to the number of subtrees. The number of parallel DFS BS EZW processors can be lowered. The simplest way to reduce the number of processors is to concatenate a number of subtrees into a single processor. If there is only one processor, we return to the sequential DFS BS EZW architecture. For ease of implementation, we assume that the number of subtrees to each processor are equal. Additional control circuitry is required if some processors process less subtrees than other processors. There is no difference in the architecture or implementation of a DFS BS EZW subtree processor for processing one single subtree or two or more subtrees in sequence.

Let N be the number of subtrees and M the number of processors. Each processor then processes N / M number of subtrees. If $N / M = 1$, then we have maximum parallelism. On the other hand, if $N / M = N$, i.e. $M = 1$, we have the sequential DFS BS EZW architecture. The number of temporary storage words for the information kept in the overall parallel processor is $N \times NC$ where NC is the number of coefficients in a subtree. This number is independent of the number of subtree processors for the storage of SIGN and MSIG. Once the subtrees are partitioned and allocated to the processors, the processing of the subtrees by each processor is exactly similar to the case of the sequential architecture. An important aspect of the parallel processing of the subtrees is on how to multiplex the output from each subtree together back into a single bit stream.

Let SAQ_{ik} be the output SAQ symbol and MAP_{ik} the output MAP symbol of the ith subtree during the kth clock cycle in the SAQ pass and the MAP pass respectively. The SAQ pass and MAP pass corresponds to First Stage and Second Stage processing respectively. The basic operation to multiplex the output bits from the subtrees is to take the output symbols from each subtree in a round robin fashion. Due to the scanning of the coefficient positions one by one in the SAQ pass to generate the SDF bits and in the same process to output the SAQ bits, there are clock cycles that do not produce an output bit from a subtree.

Another complication is to indicate if the output from a subtree is valid so that it can be read and the output bit can be collected. In the DFS BS architecture, this information is readily available. When a coefficient position is scanned during the SAQ pass, the $MSIG_{ik}$ bit is read. If $MSIG_{ik} = 1$, then it indicates that the output from the parallel processor is valid and is to be collected for this subtree. Each clock cycle in the parallel processors generate a maximum of N bits from their outputs. The multiplexer needs to read each of the $MSIG_{ik}$ bits and collect the output bits for those outputs where $MSIG_{ik} = 1$. In the MAP pass, a similar approach can be used to collect the MAP_{ik} symbols. In this case, the multiplexer uses the NZF_{ik} bit which can be output together with the MAP symbols. If $NZF_{ik} = 1$, the output from the processor processing the ith subtree is valid. In this case, two bits are collected from the output for the MAP symbols of this subtree. Figure 3.17 shows a schematic diagram of the overall structure of the parallel DFS BS EZW architecture.

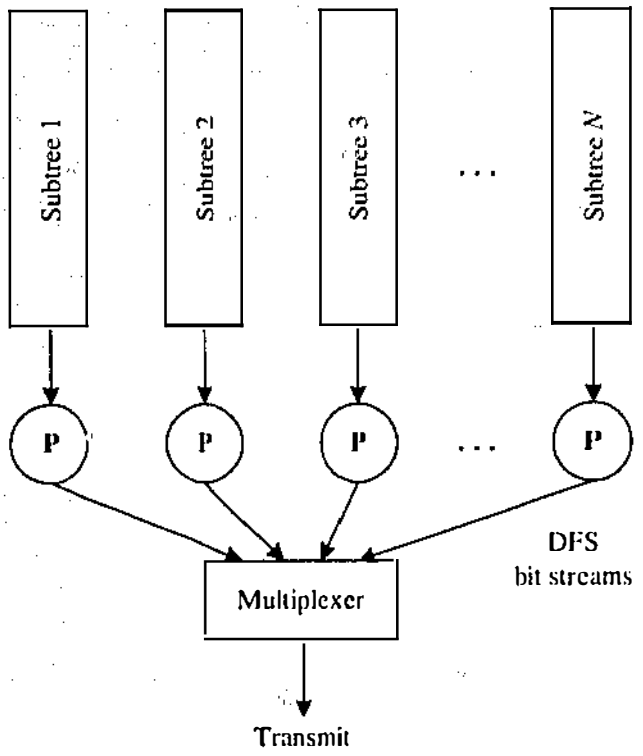


Figure 3.17 Structure of the parallel DFS BS EZW architecture

3.3.2 Output Buffering Requirements

Since a subtree does not always output a symbol during a clock cycle, a buffer at the output of the multiplexer is required to overcome the asynchronous nature of the overall output bit stream if synchronization of the output bit stream is required for the next stage of processing. The length of the buffer depends on the statistics of how the EZW symbols are generated. A handshaking circuit between the multiplexer and each of the subtree processors can be used to prevent the buffer from overflow. A subtree processor only outputs its symbol to the multiplexer if the buffer is not full. While handshaking can be used to prevent the buffer from overflowing, underflow, where there is no output from any of the subtrees imposes a more serious problem. In the extreme case where a blank image or a blank frame of video is input to the EZW system, there will be no significant coefficients in every scan of the coefficient tree. As

a result, there will be only N MAP symbols, i.e. ZTRs generated during each scan of the coefficient tree. A blank video frame is not unusual in the case of video coding.

While in the First Stage of processing by a subtree processor, every node in the subtree needs to be searched to establish the SDF bits and to output the SAQ bits, in the Second Stage, a more intelligent scheme can be used to avoid scanning each of the nodes. Equipped with the knowledge of the ancestor-descendant relationship, the search can skip to a node that has a symbol to output. This will inevitably make the addressing scheme more complex. Another way to ease the underflow problem is to perform another kind of parallel processing. In this case, a subtree processor can be modified to accept two subtrees. While the processor is performing the First Stage of one subtree, it simultaneously performs the Second Stage for the other subtree. In addition, we can also combine the outputting of the SAQ and MAP symbols in the Second Stage. These modifications will reduce the output buffer requirements but will increase the complexity of a subtree processor.

3.3.3 Different Output Schemes

In the previous chapter, we have found that the DFS causes a slight decrease in the coding performance when compared to the raster scan method used in the original EZW algorithm. During the outputting of the MAP symbols in the Second Stage, instead of scanning the storage linearly from top to bottom using the DFS scanning scheme, an address generator can be incorporated to generate the corresponding raster scan address in the DFS storage. In this way, the order of outputting MAP symbols can be generated according to the raster scan scheme. For the SAQ symbols, if reordering is performed, the order in which they are output from the subtree processors is not significant. Since

the original EZW algorithm was reported, there have been several improvements proposed to increase its coding performance. The proposed improvements make use of modified and additional MAP symbols that have different lengths of binary representations ranging from one to three bits. To implement the improved algorithms, the multiplexer can be modified to read different number of bits from the subtree processors with an increase in its implementation complexity.

Another more complex modification to the multiplexer can be in the way it collects the output bits from the subtree processors. Instead of a round robin fashion, the collection can be guided by some heuristics resulting in schemes that are similar in principle to the set partitioning search strategy used in the SPIHT algorithm. One possible heuristic would be to collect more bits from subtree processors with a higher number of significant coefficients as this may imply that the information contained in these subtrees may be more likely to be significant than the information contained in other subtrees. To further increase the options for bit collection, each subtree processor can be further split into three orientation processors corresponding to the three higher frequency orientation branches of a subtree. Figure 3.18 shows the structure of the heuristic-guided parallel architecture with the HL, LH and HH orientation processors. This structure allows for different heuristics to guide the collection of bits for the various frequency orientation branches. A simple way to deal with the LL root coefficient is to always imply that SDF = 1 so that the search will descend to the three orientation branches. In the same way, each orientation processor can be further split into another three sub-branch processors if more collection options are required. The DFS allows the partitioning of a coefficient tree into as many or as few sub-branches as required by the application.

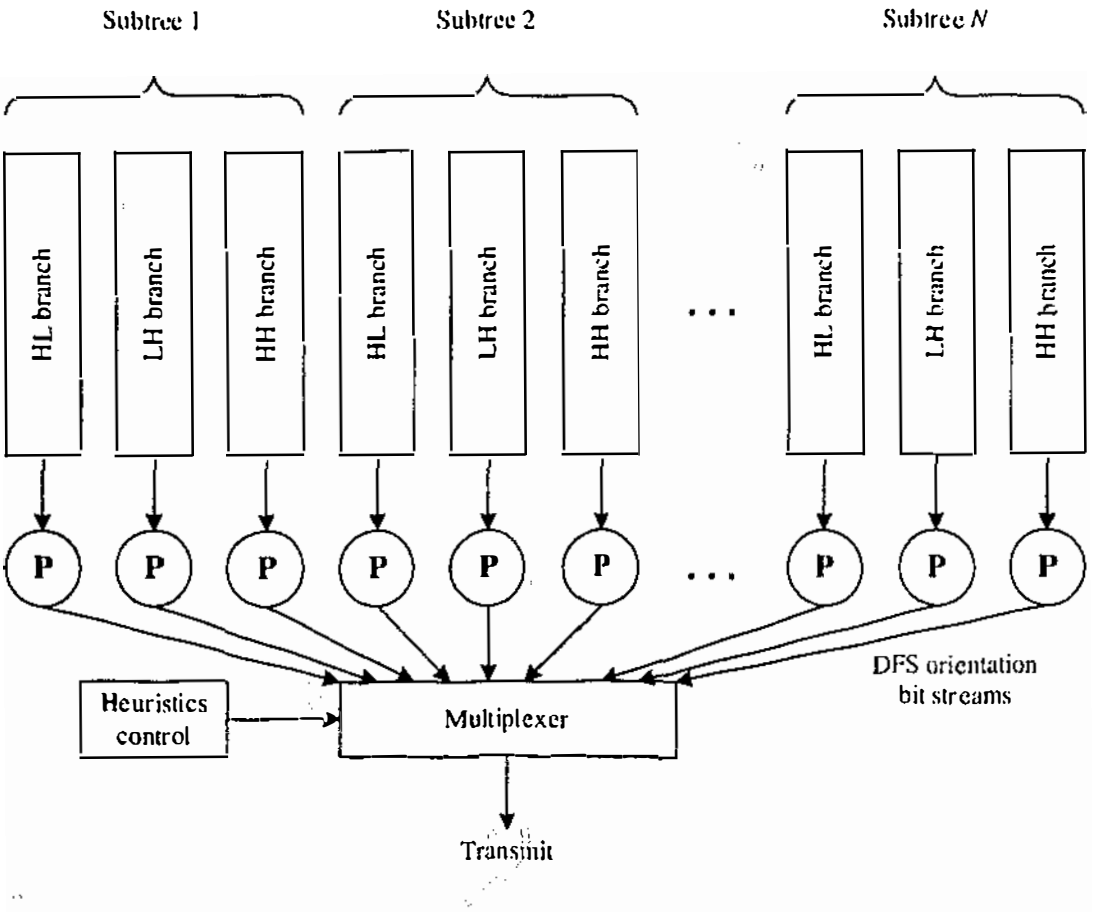


Figure 3.18 Structure of the heuristic-guided parallel architecture

3.4 Conclusions

In this chapter, we have presented new architectures for the implementation of EW algorithms based on the DFS representation of the coefficient tree. The DFS performs a natural partitioning of the coefficient tree into subtrees which can be processed independently or in parallel and provides an efficient scheme to determine ancestor-descendant relationships in the tree structure. We have presented two new architectures and focused the discussion on the issues of hardware simplicity and flexibility to implement algorithm variations.

For the first issue of simplicity, we have presented the DFS BS EZW architecture which is fast and uses minimal storage and simple processing. The feature of the bit

stream architecture is that the coefficient magnitudes are not kept in the processor but are discarded after use. By the DFS method, it is possible to process the coefficient bits without direct involvement of its ancestors or descendants. The architecture uses two stages of processing corresponding to two scans of the coefficient tree. A counter is used to keep track of which level is being searched, simple logic circuits are used for various functions and another counter is used to generate the address to access the information in the storage. While the SIGN and MSIG bits need to be kept for each node in the coefficient tree, the information in SIG and SDF is only used for one scan of a subtree and their storage can be reused for processing of a new subtree.

For applications which only require hardware encoding capability such as digital cameras and satellite imagers, the DFS BS encoder together with a suitable DWT is sufficient to provide a hardware encoder solution. For applications which require both hardware encoding and decoding capabilities such as video coding, a DFS BS decoder is required as well. We have discussed the decoder architecture and have identified a complexity for the architecture due to the compaction of the bit stream by channel coding. We have proposed two approaches for the decoder. The first approach is to use an input buffer to restore back the uncompacted bit stream. The issue here is the size of the input buffer required. If the required buffer size is small then this approach looks promising. The size of the buffer would be large if the number of ZTR symbols received is high but could be smaller if the number of ZTR symbols received is lower. The optimal size of the input buffer can be determined by statistical simulations. The second approach is to use a more complex scheme to perform the decoding where each bit which is output from the decoder is accompanied by an address which informs the wavelet processor of the position of the coefficient in the tree. This approach would be

suitable for architectures where the DWT and TS processors share the same storage and processing resources.

For the second issue of flexibility to implement algorithm variations, we have focused the discussion using the parallel DFS BS EZW architecture. The architecture makes use of the DFS BS EZW processor to process a subtree or a concatenation of several subtrees to produce intermediate information in the DFS format in parallel. The proposed architecture is independent of the subtree processor in that another processor implementing other MAP coding schemes can be used instead. Once the intermediate information has been generated in the First Stage, in the Second Stage, the MAP symbols can be output in the DFS format or another format by using a more complex multiplexer. Rather than the linear addressing scheme used in the basic DFS subtree processor, a generator to produce the addressing scheme corresponding to a different search method can be incorporated. The proposed parallel architecture provides a flexible scheme for different degrees of parallelism and can be easily modified to implement the Improved EZW algorithm or algorithms similar to the SPIHT algorithm.

When multiplexing the output from each subtree processor back to a single bit stream, the problem of overflowing the output buffer is addressed by the use of handshaking. A more serious problem is the underflow of the output buffer where there is very low output from a pass of the coefficient tree which will lead to an asynchronous output situation in the overall output bit stream. The underflow situation can be eased by the use of more complex addressing schemes in the Second Stage of processing to produce the output symbols.

The sequential and parallel DFS BS approaches give a diversity of implementation options and coding performances to meet the requirements for different applications. For example, a videophone application may use a sequential DFS BS architecture and be sufficient to meet the required performance. For applications which require high performance such as HDTV, the parallel DFS BS architecture gives faster processing and higher performance due to better output collection methods. Parallel processing is used not only to increase the speed of processing but also to achieve higher performance.

We have identified a number of complexities which can be solved by having a single structure for the DWT and TS processors where storage and processing are shared. The SIGN and MSIG bits required for encoding can be stored in the DWT processor. The input and output buffering requirements for the TS processor can be minimized by using a scheme whereby the DWT processor communicates with the processor for each bit which is input or output from the processor. In the next chapter, we will discuss different hardware approaches for implementing the joint DWT-TS system.

Chapter 4

Hardware Implementation of the DFS BS SPIHT System

4.1 Introduction

The hardware implementation of embedded wavelet (EW) algorithms would require two sets of decisions to be made. The first decision is to select the algorithm variation to be implemented and the second decision is regarding the hardware platform for implementation. There may also be a need to consider the suitability of the hardware for the selected algorithm variation. Certain variations in the algorithm may incur much more implementation complexity than other variations because of constraints specific to the hardware technology. Figure 4.1 shows the variations in EW algorithms for hardware implementation. The variations are divided into the four categories of tree structure, search strategy, coding scheme and symbol stream. We have already discussed these variations in Chapter 2. In this chapter, we select a particular variation for hardware implementation. The variation we have selected is highlighted in the figure.

Tree structure	Search strategy	Coding scheme	Symbol stream
EZW	***DFS***	EZW	Alternating
	BFS	Improved EZW	
SPIHT	Set partitioning	***SPIHT***	***Mixed***

Figure 4.1 Variations for EW algorithms

From the discussion in the previous chapters, we have found that the EZW tree structure and the DFS tree searching strategy are suitable for hardware implementation. The former because it treats the root nodes as any other tree node and has a more

suitable ancestor-descendant relationship in the tree structure. The latter because it allows for the bit stream architecture where the coefficient magnitude bits do not have to be stored and are processed as they flow through the processor. We have selected the SPIHT coding scheme because it gives the highest coding performance and the mixed symbol stream because it minimizes the input and output buffering requirements.

The SPIHT algorithm can be described as consisting of discrete wavelet transform (DWT), tree searching (TS) and arithmetic coder (AC) modules. As such, the implementation of the full SPIHT algorithm may require up to three separate processors to perform the functions of the DWT, TS and AC. An advantage of the SPIHT algorithm over the EZW algorithm is that the output bit stream of the former is compact and arithmetic coding does not significantly increase the algorithm performance. By not requiring the AC module, the implementation of the SPIHT algorithm requires two processors. One processor performs the DWT/IDWT and another processor performs the TS/ITS. The DWT module is an essential component in wavelet coding systems. Several hardware architectures have been reported for the DWT for implementation in VLSI [50], [51], [52], field programmable logic devices (FPLDs) [55] and DSP processors [56]. The TS module is specific to architectures for embedded wavelet coding. The TS architectures can be broadly classified according to the way it performs searching of the wavelet trees: breadth-first search or depth-first search (DFS).

We investigate the feasibility of a single chip device to implement the DFS BS SPIHT system. A single chip solution for the DFS BS SPIHT system is a very promising solution which has low cost and high performance for video coding applications in general and for portable video communicators in particular. The work presented in this chapter gives three advantages for the hardware implementation of

video coding systems. The three advantages correspond to the three design levels of algorithm, architecture and implementation. At the algorithm level, the DFS SPIHT algorithm gives several advantages over DCT-based coding schemes used in the current JPEG [57] and MPEG [58] coding standards. In addition to high coding performance which is very close to that of the original SPIHT algorithm, the DFS SPIHT algorithm maintains the natural features of EW algorithms like bit stream scalability and the ability to meet a target bit rate exactly. At the architecture level, the DFS SPIHT algorithm can be easily implemented using low complexity bit stream architectures.

In this chapter, we will look at the advantages of the DFS BS SPIHT architecture for implementation. We will discuss how the bit stream TS/ITS processor can be efficiently combined with a DWT/IDWT processor to form a single architectural structure for hardware implementation. We will discuss two approaches for hardware implementation. The first approach uses a conventional memory bank approach and the second approach uses a futuristic smart pixel (SP) VLSI approach. We apply the bit stream DFS BS SPIHT EW processor for the memory bank approach and the parallel DFS BS SPIHT EW architecture for the SP approach. For the various approaches, we will consider two implementation issues. The first issue is the interfacing of the DWT/IDWT processor with the TS/ITS processor. The second issue is the distribution of the processing and storage requirements amongst the DWT/IDWT processor and the TS/ITS processor to achieve an optimal system.

We present a memory bank implementation of the DFS BS SPIHT system which can be switched to perform encoding and decoding on the same device. The input into the system during encoding is a stream of image pixels and the output is an encoded bit stream ready for transmission. The system produces the reconstructed image data from

the received bit stream during decoding. The proposed system uses two memory banks for processing: coefficient memory bank and subtree memory bank. To reduce storage requirements, the coefficient memory bank is used at different stages of processing to store the image pixels or wavelet coefficients. In the encoding stage, the coefficient memory bank contains the image pixels initially. After the DWT is performed, the memory bank contains the wavelet coefficients. During decoding, the coefficient memory bank stores the reconstructed wavelet coefficients and after the IDWT contains the decoded image pixels.

This chapter is organized as follows. In Section 4.2, we discuss the DFS SPIHT algorithm in relation to the original SPIHT algorithm. We discuss the memory bank and SP VLSI approaches for hardware implementation in Section 4.3 and Section 4.4 respectively. In Section 4.5, we present a memory bank implementation of the DFS BS SPIHT system using SYNOPSYS simulation and synthesis tools. We give conclusions in Section 4.6.

4.2 DFS SPIHT Algorithm

The SPIHT algorithm maintains three ordered lists to store the significance information. The three lists are the list of insignificant sets (LIS), list of insignificant pixels (LIP) and list of significant pixels (LSP). The SPIHT algorithm as reported in [17] is shown in Figure 4.2. The algorithm uses the following functions to indicate the significance of the coefficients or its descendants:

- $S_n(i,j)$: significance of the coefficient;
- $S_n(D(i,j))$: significance of the descendants of the coefficient;

- $S_n(L(i,j))$: significance of the descendants of the coefficient with the exception of its immediate children

where (i,j) is a coordinate in the LIS, LIP or LSP. The SPIHT algorithm poses two complexities for implementation:

1. The search order of the coefficient tree is according to the set partitioning strategy. This requires the coefficients to be processed in the order indicated in the LIS;
2. The size of the LIP would decrease from a maximum and the size of the LSP would increase from a minimum.

Complexity (1) causes irregularity in processing the coefficient tree where the order of processing is not fixed but depends on the LIS. Complexity (2) would require either variable storage schemes or a worst-case of allocating the maximum memory required for both LIP and LSP.

To overcome the two Complexities, we propose the following modifications:

1. The depth-first search is used instead of the set partitioning search strategy. This permits a regular processing of the coefficient tree according to the DFS traversal order. Moreover, since the order of processing is fixed, the LIS need not be kept, thus reducing the memory requirements.
2. Coefficients which have been found to be significant are not taken out of the tree but are marked significant (MSIG). The LIP and LSP are kept together in the tree structure and the MSIG symbol indicates if the coefficient is in the LIP or LSP.

In the architecture, each node of the coefficient tree is associated with a set of binary symbols. Table 4.1 shows the MAP symbols used in the DFS SPIHT algorithm and the corresponding SPIHT significance functions. An additional symbol which is used in the DFS SPIHT algorithm is MSIG.

Table 4.1 MAP symbols for DFS SPIHT and SPIHT algorithms

DFS SPIHT	SPIHT
SIGN	Sign
SIG (Significant Coefficient Flag)	$S_n(i,j)$
SDF (Significant Descendant Flag)	$S_n(D(i,j))$
SGDF (Significant Grand Descendant Flag)	$S_n(L(i,j))$
MSIG (Marked Significant Flag)	-

Figure 4.3 shows the DFS SPIHT algorithm. The wavelet coefficients are in the sign-magnitude representation where the sign bits are stored in SIGN and the magnitude bits are stored in MAG. The number of magnitude bits used to represent the coefficient is given by n and each tree node is represented by (i). The symbol LEAF(i) indicates if the node is a leaf in the coefficient tree. The output from the algorithm contains a mixture of MAP and MAG symbols. To separate the MAP and MAG symbols for transmission would require additional storage. To avoid the additional storage, we propose a third modification:

3. The MAP and MAG symbols are transmitted as they are generated.

Figure 4.3 shows that by using the DFS and bit stream processing, the implementation of the DFS SPIHT algorithm is simplified when compared to the original SPIHT algorithm.

SPIHT Algorithm:

1) **Initialization:**

output $n = \lfloor \log_2(\max_{(i,j)} |c_{i,j}|) \rfloor$;
set the LSP as an empty list, and add the coordinates $(i,j) \in H$ to the LIP, and only those with descendants also to the LIS,as type A entries.

2) **Sorting Pass:**

2.1) for each entry (i,j) in the LIP do:

2.1.1) output $S_n(i,j)$;
2.1.2) if $S_n(i,j) = 1$ then move (i,j) to the LSP and output the sign of $c_{i,j}$;

2.2) for each entry (i,j) in the LIS do:

2.2.1) if the entry is of type A then output $S_n(D(i,j))$;

- if $S_n(D(i,j)) = 1$ then for each $(k,l) \in O(i,j)$ do:
  - output $S_n(k,l)$;
  - if $S_n(k,l) = 1$ then add (k,l) to the LSP and output the sign of $c_{k,l}$;
  - if $S_n(k,l) = 0$ then add (k,l) to the end of the LIP;
  - if $L(i,j) \neq 0$ then move (i,j) to the end of the LIS, as an entry of type B, and go to Step 2.2.2); otherwise, remove entry (i,j) from the LIS;

2.2.2) if the entry is of type B then

- output $S_n(L(i,j))$;
- if $S_n(L(i,j)) = 1$ then
- add each $(k,l) \in O(i,j)$ to the end of the LIS as an entry of type A;
- remove (i,j) from the LIS.

3) **Refinement Pass:**

for each entry (i,j) in the LSP, except those included in the last sorting pass (i.e., with same n), output the nth most significant bit of $|c_{i,j}|$;

4) **Quantization-Step Update:**

decrement n by 1 and go to Step 2.

Figure 4.2 SPIHT algorithm

DFS SPIHT Algorithm:

1) **Initialization:**

output n and set MSIG(i) to 0 for all tree nodes

2) **Coding and Refinement Pass:**

2.1) for each node (i) in the coefficient tree do:

2.1.1) if MSIG(i) = 1 then output MAG(i,n)

2.1.2) if MSIG(i) = 0 then

- output SIG(i)
- if SIG(i) = 1 then output SIGN(i) and set MSIG(i) = 1
- if LEAF(i) = 0 then output SDF(i) and SGDF(i) if SDF(i) = 1

3) **Quantization-Step Update:**

decrement n by 1 and go to Step 2

Figure 4.3 DFS SPIHT algorithm

4.3 Memory Bank Implementation

Figure 4.4 shows the memory bank DFS BS SPIHT structure. The encoder and decoder consist of four modules and two memory banks. The memory banks are used at different stages to perform encoding and decoding. During encoding, the coefficient memory bank initially contains the image pixels. The DWT processor operates on the image pixels and transforms them into their corresponding wavelet coefficients. The coefficients are stored in their sign-magnitude representation. The SIGN, MAG and MSIG bits for a subtree are shifted out of the memory bank in decreasing order of their bit planes. The DWT-to-DFS converter converts the MAG bits and their corresponding SIGN and MSIG bits into the DFS format for input into the First Stage TS encoder. The converter can be implemented using a look-up table in ROM. The First Stage TS

encoder calculates the SIG, SDF and SGDF bits and stores the SIG, SIGN, SDF, SGDF and MSIG bits in the subtree memory bank. The Second Stage TS encoder reads the bits out from the subtree memory bank and outputs the MAP for non MSIG bits and outputs the SAQ for MSIG bits. During decoding, the transmitted bit stream is input into the Second Stage TS decoder which decodes the MAP and SAQ bits into the corresponding SIGN and MAG bits. The decoded SIGN and MAG bits are converted from the DFS format back to the DWT format by the DFS-to-DWT converter and stored in the coefficient memory bank. The IDWT processor then operates on the coefficients to restore the image pixels.

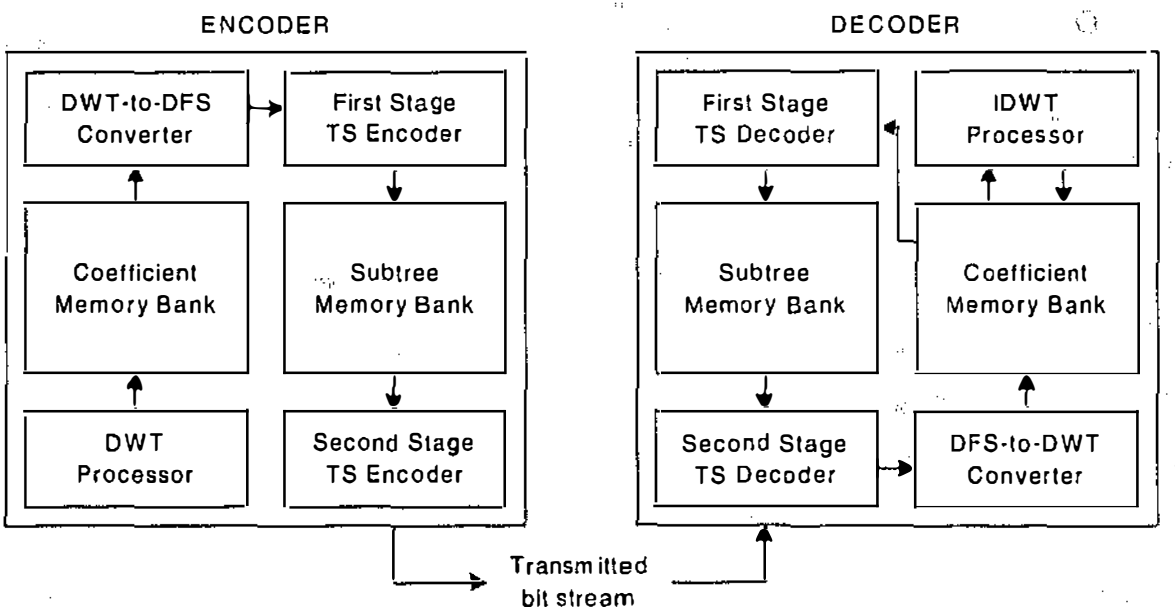


Figure 4.4 Memory bank DFS BS SPIHT structure

The First Stage TS decoder and the subtree memory bank are not essential for the decoding process. They are used to simplify the processing requirements for the Second Stage TS decoder. One requirement for the Second Stage decoder is to determine if coefficients are already MSIG. MAP bits and SAQ bits are decoded from the transmitted bit stream for non-MSIG coefficients and MSIG coefficients

respectively. Figure 4.5 shows a coefficient tree with a MSIG node. The condition SDF = 0 implies that there are no MAP bits to be decoded for the coefficients. However, this does not mean that there are no SAQ bits to be decoded. The decoder would still have to decode SAQ bits for MSIG coefficients even though its ancestor nodes have SDF = 0. In order to do this, the Second Stage decoder would have to scan the coefficient memory bank linearly to determine MSIG coefficients.

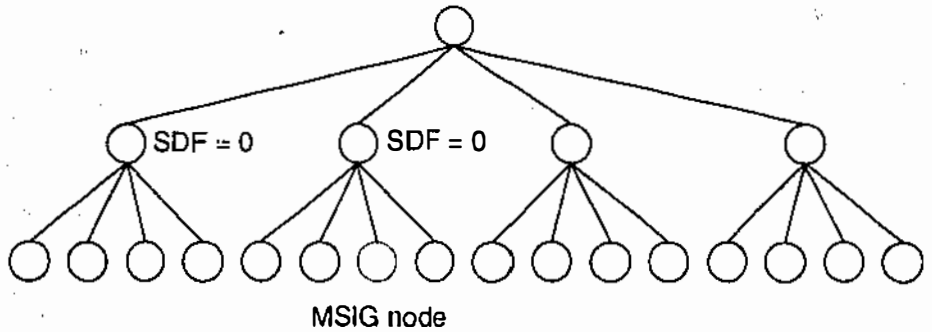


Figure 4.5 Coefficient tree with MSIG node

The purpose of the First Stage decoder is to read the coefficient memory bank and use the subtree memory bank to store the MSIG information in a tree structure similar to the structure used in the encoding process. In this case, we use the SIG and SDF bits to determine the MSIG tree. MSIG coefficients have SIG = 1. The condition SDF = 0 implies that there are no MSIG descendant nodes in the branch. The Second Stage decoder searches the tree to determine MSIG coefficients. Using the subtree memory bank to store the MSIG tree does not increase the hardware complexity significantly because the memory storage is required for encoding. It does however simplify the processing requirements for the Second Stage decoder for determining MSIG coefficients.

4.3.1 DWT/IDWT Processor

Figure 4.6 shows an 8×8 image pixel array and the DWT coefficients for three scales of subband decomposition. The subbands arise from separable application of row and column filters at each scale. In each row and column filtering, low-pass and high-pass filters are applied. The outputs of the low-pass and high-pass filters are decimated by two to give the same number of coefficients as image pixels. The filtering process is applied repeatedly on the lowest frequency subband until the required number of scales is reached.

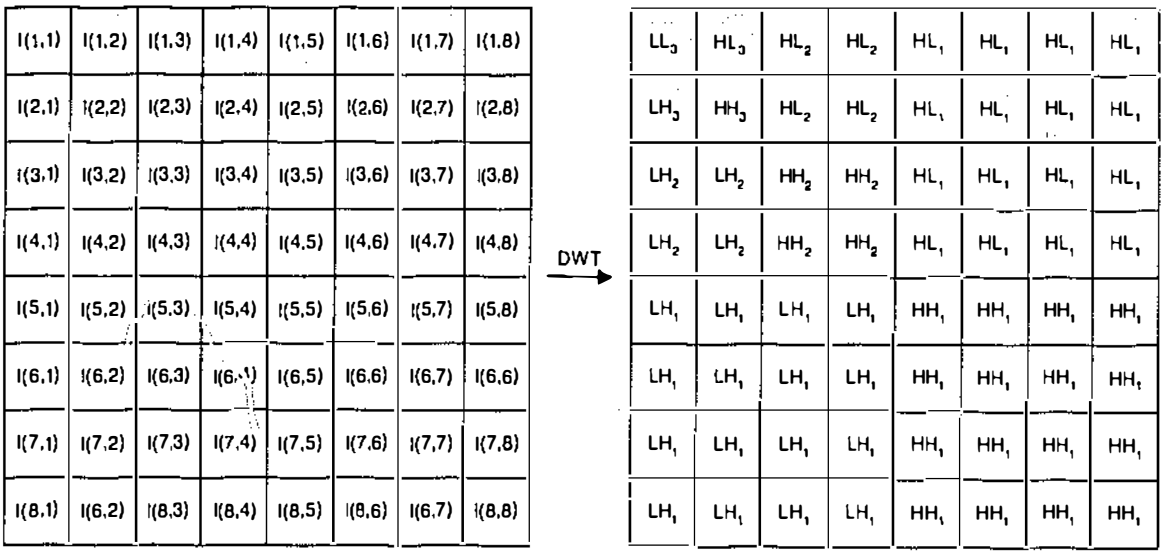


Figure 4.6 Image pixels and DWT coefficients

Figure 4.7 shows the arrangement of the image pixels in the coefficient memory bank for encoding. The image rows are arranged consecutively in the memory bank. At each stage, the coefficients pointed to by a set of memory pointers are filtered. We illustrate the scheme using three-tap filters. The scheme can be extended for higher tap filters by increasing the number of memory pointers. We define three memory pointers Back, Current and Front as shown in Figure 4.7(a).

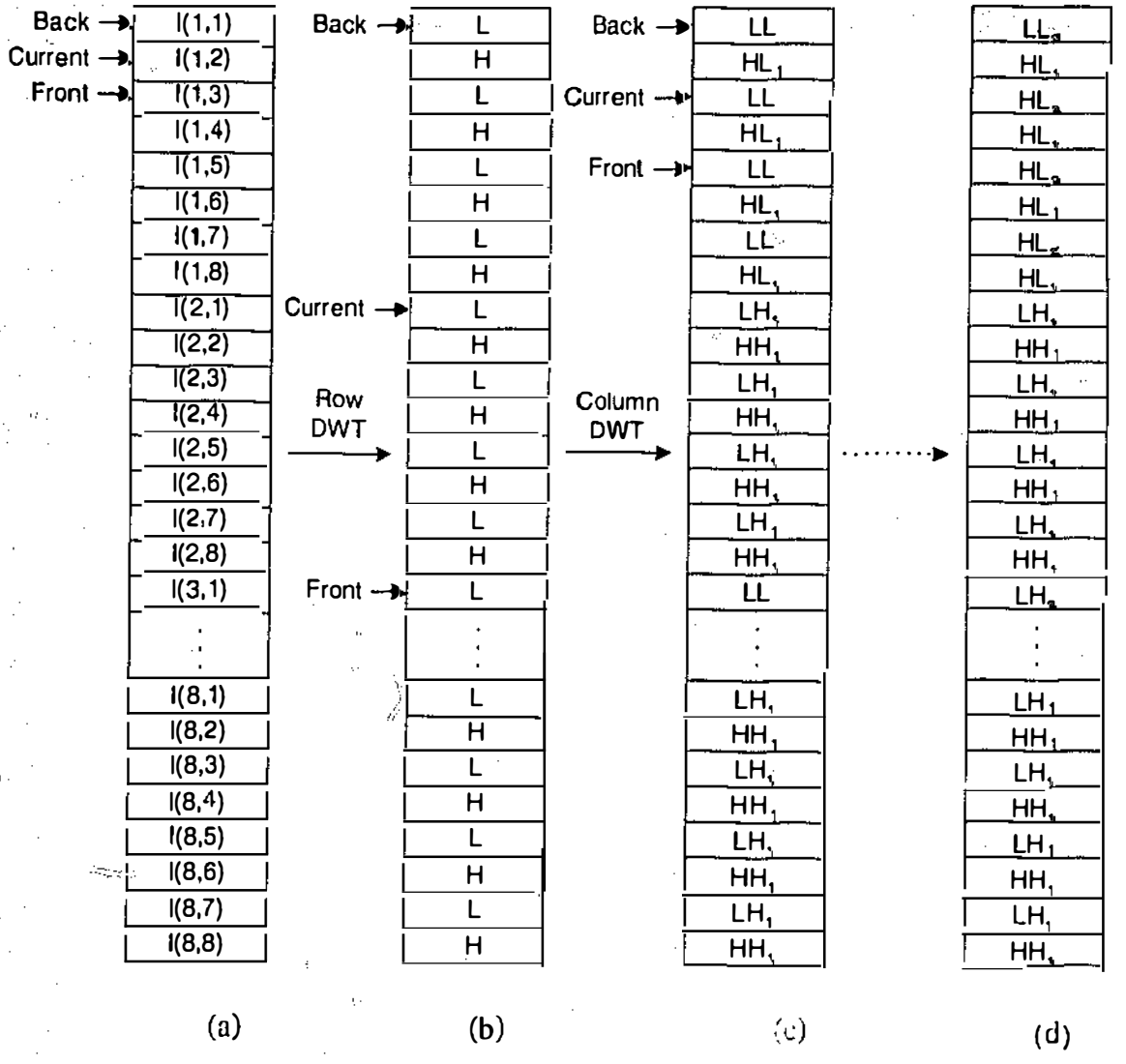


Figure 4.7 Memory bank DWT arrangement for encoding

To perform the row filtering, the memory pointers are set to consecutive locations and moved one location at a time down the memory bank. The Back and Front memory pointers differ from the Current pointer by an Offset depending on the current scale. For the S^{th} scale of subband decomposition, the Offset is given by 2^{S-1}. The memory pointers are given by:

$$\text{Back} = \text{Current} - \text{Offset} \tag{4.1}$$

$$\text{Front} = \text{Current} + \text{Offset} \tag{4.2}$$

The low-pass and high-pass coefficients are stored in alternating memory locations as shown in Figure 4.7(b).

To perform the column filtering, the memory pointers are initialized as shown in Figure 4.7(b). The pointers are given by:

$$\text{Back} = \text{Current} - \text{Offset} \times \text{Image\_Col} \tag{4.3}$$

$$\text{Front} = \text{Current} + \text{Offset} \times \text{Image\_Col} \tag{4.4}$$

where Image_Col is the number of columns in the image. For filtering at the row and column boundaries, symmetric extension of the image is applied. For filtering at the beginning of rows or columns, the Back pointer is set to Front and for filtering at the end of rows or columns, the Front pointer is set to Back. For each scale of subband decomposition, only the lowest frequency subband is further decomposed. After the last coefficient in a column is filtered, the Current memory pointer is set to skip the higher frequency subbands which are not required for further filtering. The memory pointer is set by:

$$\text{Current} = \text{Current} + (\text{Offset} - 1) \times \text{Image\_Col} \tag{4.5}$$

The filtering process is repeated for the number of scales of decomposition required. After the filtering, the memory bank contains the wavelet coefficients in the arrangement as shown in Figure 4.7(d).

The reverse process is performed for decoding. Initially, the memory bank contains the wavelet coefficients and after decoding it contains the decoded image pixels. Figure 4.8 shows the IDWT arrangement for the decoding process. The IDWT processing alternates between column filtering and row filtering for each scale. The operations required for the DWT encoding and IDWT decoding processes are similar. The difference is to use analysis filters for the DWT decomposition and synthesis filters for the IDWT reconstruction.

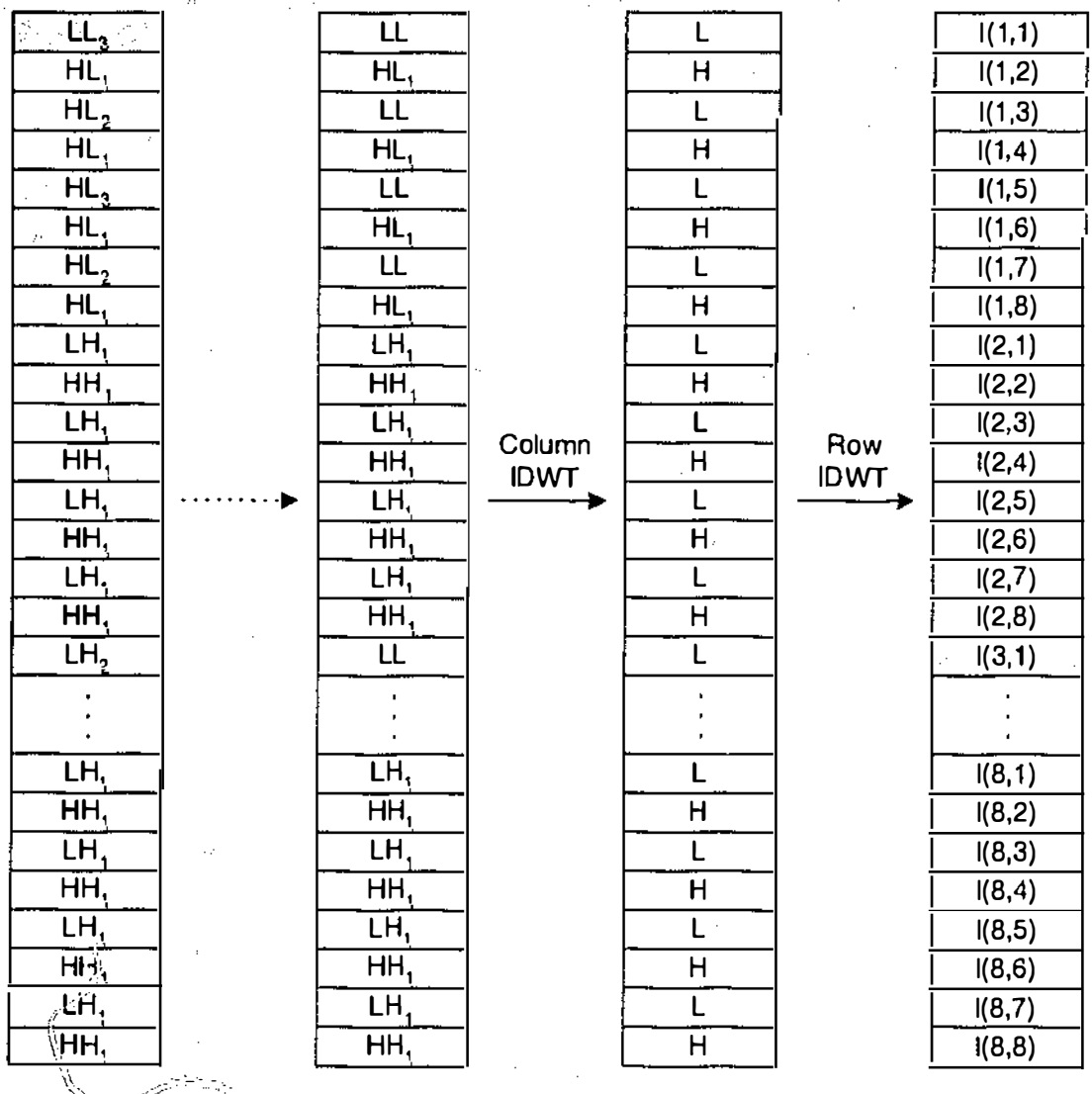


Figure 4.8 Memory bank IDWT arrangement for decoding

4.3.2 DWT-to-DFS/DFS-to-DWT Converter

For input into the TS encoder and decoder, the DWT format in the coefficient memory bank has to be converted into the DFS format. These conversions can be performed using look-up tables (LUT) implemented in ROM. The number of entries in the LUT is determined by the size of the subtree. Figure 4.9 shows the memory bank DWT arrangement for four subtrees.

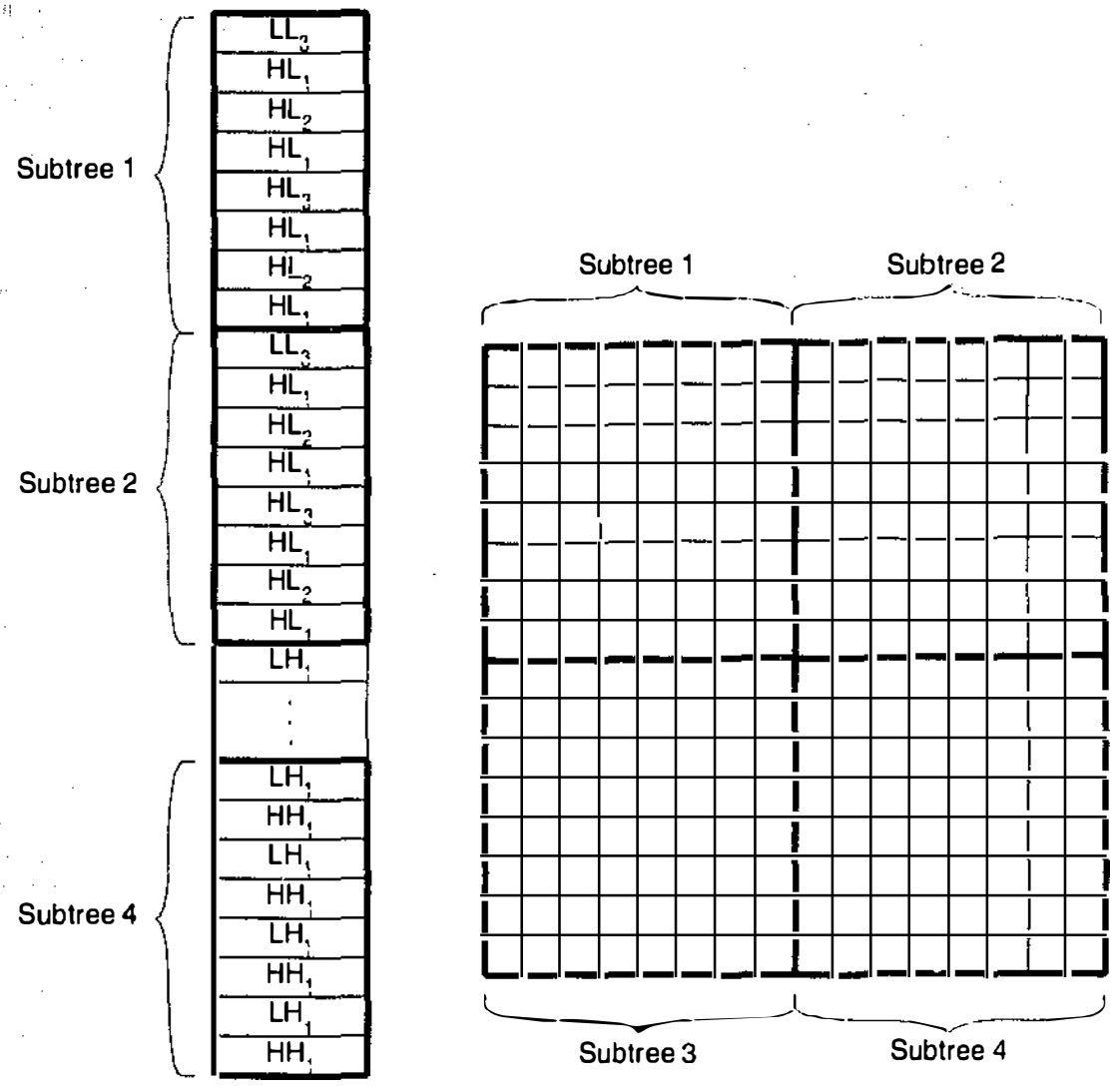


Figure 4.9 Memory bank DWT arrangement for four subtrees

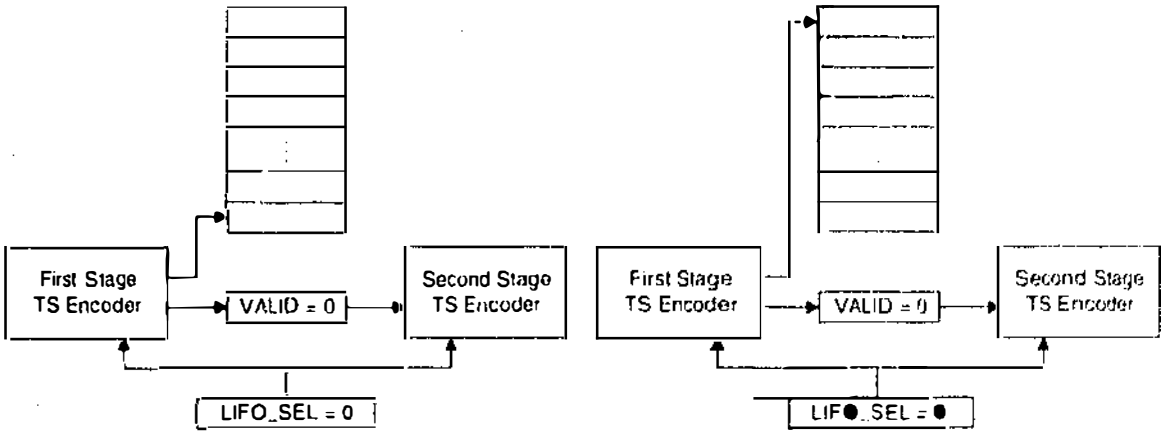
The coefficients for the first row of subtree 1 are stored in the first eight consecutive memory locations and the coefficients for the first row of subtree 2 are stored in the next eight consecutive locations. The last eight locations store the coefficients for the last row of subtree 4. The rows for each subtree are not located in consecutive locations but are dispersed throughout the memory bank. One approach to bring the subtree rows into consecutive locations is to perform a rearrangement of the coefficients in the memory bank. This approach would require additional processing to be performed. A better approach would be to leave the coefficients in the memory bank and use a more intelligent addressing scheme to bring the coefficients for each subtree together. This

approach takes advantage of the fact that although the subtree rows are dispersed throughout the memory bank, the number of memory locations from one subtree row to the next is always the same for a particular image size.

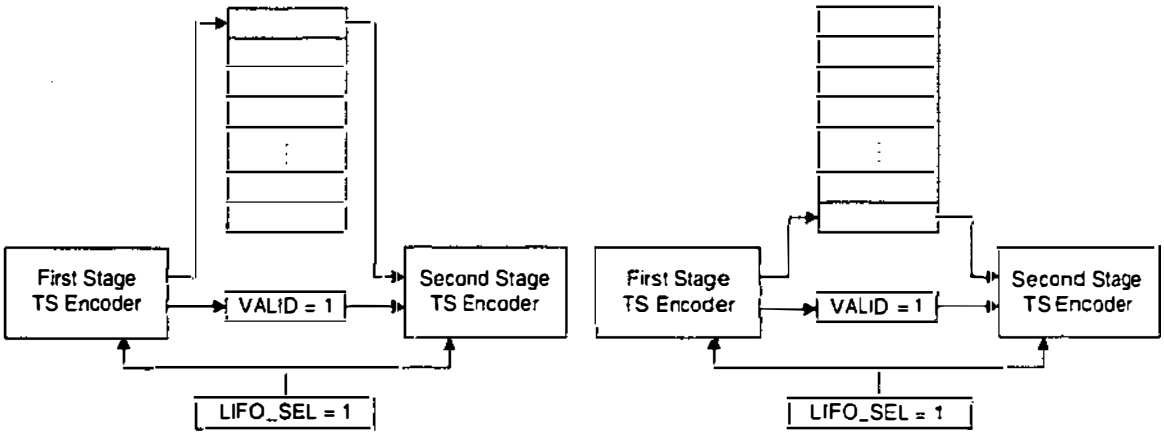
4.3.3 TS/ITS Processor

The architectures for the TS and ITS processors have been described in Chapter 3. In this section, we focus our discussion on the handshaking mechanism between the First Stage and Second Stage processors to access the subtree memory bank. The handshaking mechanism between the First Stage and Second Stage processors is similar to the producer-consumer problem with a difference. The difference is that the First Stage processes from the leaves of the tree to the roots whereas the Second Stage processes from the roots to the leaves. This requires a LIFO memory storage.

We define two signals VALID and LIFO_SEL to perform the handshaking between the processors. The VALID signal indicates to the Second Stage processor that there is an item of data which is ready for processing. The VALID signal is set to 0 initially and is subsequently set to 1 after the First Stage processor has finished processing the first subtree. The LIFO_SEL signal indicates the direction to scan the memory bank. When LIFO_SEL = 0, the memory bank is scanned from the bottom to the top of the memory bank and when LIFO_SEL = 1, the memory bank is scanned from the top to the bottom of the memory bank. The LIFO_SEL signal is toggled after each subtree has been processed. Figure 4.10 shows the handshaking mechanism between the First Stage and Second Stage processors using the VALID and LIFO_SEL signals.



(a) VALID = 0 and LIFO_SEL = 0



(b) VALID = 1 and LIFO_SEL = 1

Figure 4.10 Handshaking mechanism between First and Second Stage processors

Initially, the VALID signal is set to 0 and the Second Stage processor doesn't operate. The LIFO_SEL signal is set to 0 and the First Stage processor begins storing data from the bottom of the memory bank as shown in Figure 4.10(a). When the data reaches the top of the memory bank, the VALID signal is set to 1 and the LIFO_SEL signal is toggled to 1 as shown in Figure 4.10(b). This indicates to the processors to store and read the data beginning from the top of the memory bank. For the next subtree, the LIFO_SEL signal is toggled back to 0.

4.4 Smart Pixel VLSI Implementation

In an image or video coding system, the capture, digitizing and coding of an image are normally performed by different modules. Recently, a smart pixel (SP) approach has been proposed to perform the capture and digitizing of an image in the same device [59]. In addition, a two-dimensional DWT is also embedded in the device. The proposed device will also have the capability to perform a two-dimensional IDWT and to display the resulting image. Our objective is to design a TS processor which can be merged into the proposed SP array and which will perform coding and possibly decoding using as many components as possible by switching the processor to encode or decode as required by the SP array. We will mainly describe the encoding part of the processor.

4.4.1 Structure of a Smart Pixel

The SP architecture is an array processor architecture which combines image capture and processing on a single chip. An architecture to perform the DWT within the SP array has been reported [59]. The implementation of a two-dimensional DWT consists of an array of smart pixels. Except for those at the edge of the array, each smart pixel is connected vertically and horizontally to its four nearest neighbours creating a mesh structure of processing elements. Figure 4.11 shows the structure of a smart pixel for encoding. The sensor is responsible for capturing the light at this pixel position. After the light intensity is converted to digital form and stored in the shift registers, the DWT circuitry in conjunction with those in the other pixels perform a DWT on the captured image. Figure 4.12 shows a block diagram showing the components of a complete SP DFS BS SPIHT system for image coding. The interface circuit shifts out the

information of the coefficients in the DWT format to that of the DFS bit stream format required by the TS processor. The success of the combined system depends on the use of the storage in the SP so as to minimize storage requirements in the TS processor and on shifting as much processing as possible from the TS processor to the SP array. This would require additional circuitry to be incorporated into each smart pixel to help in performing the coding.

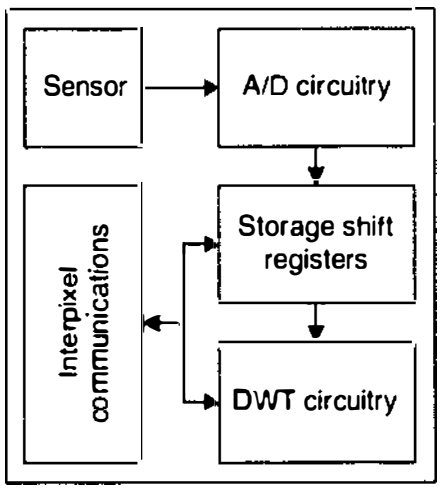


Figure 4.11 Smart pixel structure

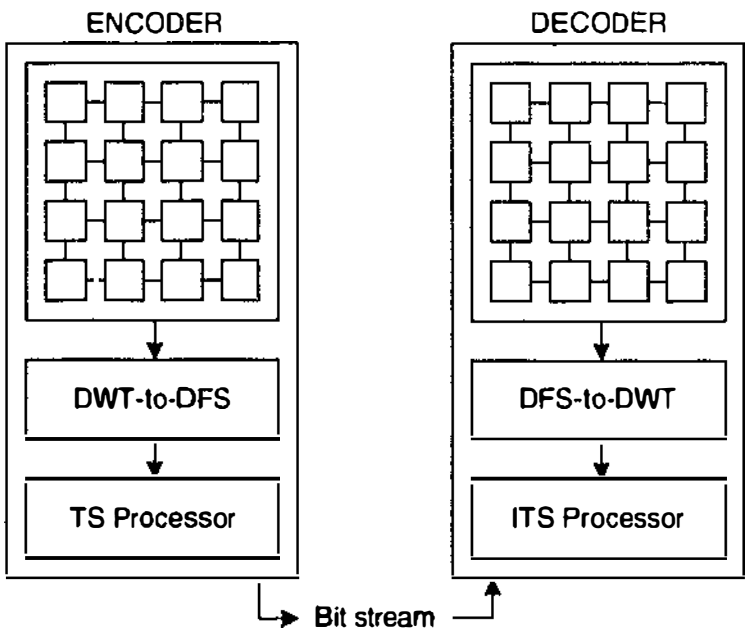


Figure 4.12 SP DFS BS SPIHT system

4.4.2 Additional Circuitry in Smart Pixel

Figure 4.13 is a block diagram showing the inside of a SP with the additional circuit for the generation of MSIG and the control for shifting out the MAG, SIGN and MSIG bits. Register MAG_BITS stores the magnitude of the resulting coefficient after a DWT has been performed on the captured image using the DWT part of the SP, while SIGN stores the sign bit. A shift register is considered to be connected through the smart pixels in each column for the purpose of shifting out the coefficient information. Each pixel occupies three bits of the shift register. The first bit is always the current MSIG bit.

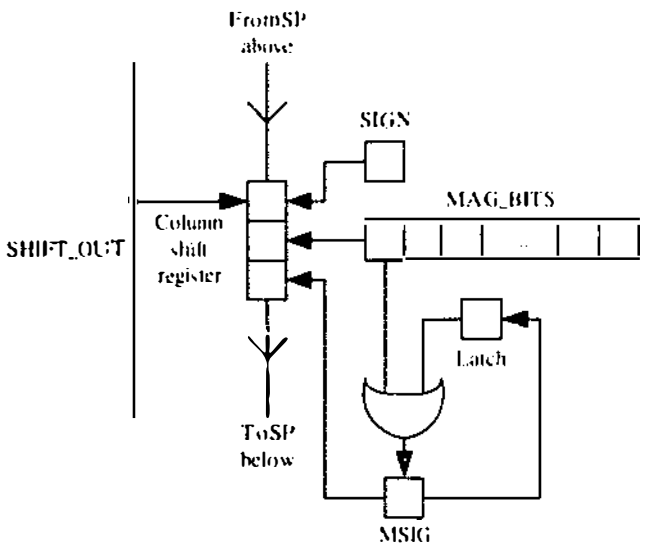


Figure 4.13 Coding component in a SP

The other part of the additional circuit is for the calculation and updating of MSIG at the end of each pass of the coefficient tree. The MSIG bit is updated according to the following equation:

$$MSIG_p = MAG_1 \text{ OR } MAG_2 \text{ OR } \ldots \text{ OR } MAG_p \tag{4.6}$$

where MAG_p is the magnitude bit of the coefficient in the pth pass of the tree with MAG_1 being the most significant bit of the coefficient. The updating circuit consists of

an OR gate with the current MSIG and MAG as the inputs. The result is latched and is written into MSIG at the end of the current pass. The contents of the register MAG_BITS are also shifted to the left. A global control signal, SHIFT_OUT is used to indicate the different events involved in the shifting of the three bits for the coefficients, the updating of MSIG and the left shifting of the bits in MAG_BITS. The actual shifting of the bits is controlled by the global clock signal. A local circuit in each SP can be designed to determine the events by counting the occurrences of the change of SHIFT_OUT.

4.4.3 DWT Interface and TS Processor

Figure 4.14 shows a block diagram of the SP interface and the TS processor. Also shown in the figure is the arrangement of the coefficients in the SP array for one subtree as a result of a three scale DWT decomposition. The arrangement of the coefficients in the array is according to the nucleic scheme [59]. By the nucleic scheme, a subtree is always located in the SP array in this manner. Each subtree is divided into four quadrants. A column of a quadrant is shifted out first followed by another column and so on. When a column is shifted, those bits of the same column in the quadrants above will be shifted to the lower quadrants. The signal SHIFT_OUT controls the shifting of a column. Three modulo four counters are used to track which quadrant the shifted out information belongs to as well as the coordinates of the original positions of the coefficients. This information is needed by the orientation demultiplexer that distributes the shifted out information to the orientation branch processors as well as by the First Stage TS processor.

The original DFS algorithm searches a subtree in such a way that the coefficients belonging to one orientation is searched completely before the search scans the part of the subtree belonging to another orientation. Due to the nature of shifting out of the bits of the coefficients in the SP array, information for the three orientations arrives at the TS processor in parallel. Therefore, the DFS is modified further to break down a subtree into three orientation branches and a root node. Each of the three orientation branches is searched in the depth-first manner but the search is performed in a pseudo parallel way in the First Stage of the processing. The search of the three orientation branches will finally merge back to the root node. After the First Stage, all the information about a subtree is stored inside the processor for the Second Stage of processing. The temporary storage for the root node and the three orientation branches will be reconfigured back into a single storage structure as defined by the DFS representation. This storage structure is then scanned from top to bottom to generate the MAP and SAQ symbols according to the DFS.

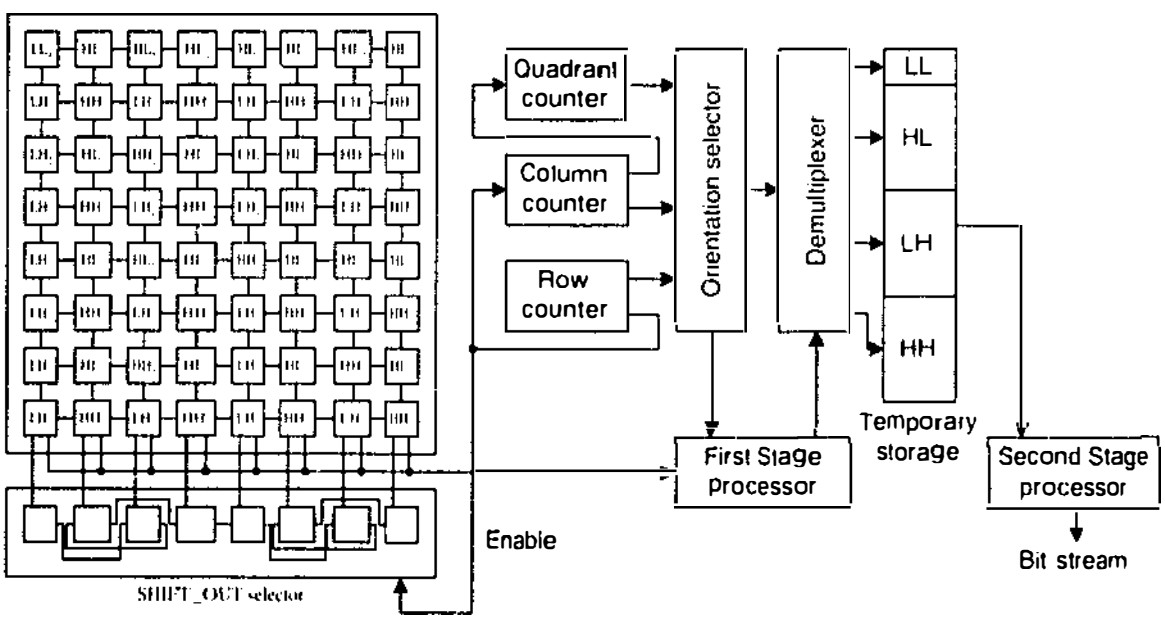


Figure 4.14 DFS BS TS processor and SP interface

4.5 Simulation Results

In this section, we present simulation results for a memory bank implementation of the DFS BS SPIHT system. For an initial investigation, the LS encoder consisting of the First Stage and Second Stage encoders and the subtree memory bank were simulated for 8×8 pixels using VHDL [60], [61] and synthesized using SYNOPSYS Behavioral [63] and Design Compilers [64] to determine its architecture complexity. The synthesized circuit was verified using the test data in [16]. We used the LSI_10K technology library supplied with SYNOPSYS with a clock period of 100 ns. The encoder was synthesized using a total cell area of 5609 gates. Next, we simulated and synthesized the DFS BS SPIHT system for 16×16 image pixels. For the DWT and IDWT, we used three scales of subband decomposition and reconstruction using the three-tap wavelet filters as described in [65]. These filters were selected because the filtering can be implemented using shifts and adds. We used 11 bits for the width of the coefficient memory bank. The DFS BS SPIHT system was synthesized using a total cell area of 61531 gates. Figure 4.15 shows a top level schematic of the system.

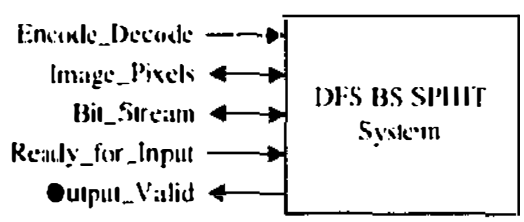


Figure 4.15 Schematic of DFS BS SPIHT system

Table 4.2 shows the characteristics of the data signals. The Encode_Decode input signal determines if the device is set to perform encoding or decoding. The signal is set to 0 for encoding and 1 for decoding. During encoding, Image_Pixels contain the input data to be encoded and Bit_Stream contain the output bit stream to be transmitted.

During decoding, Bit_Stream contain the input bit stream to be decoded and Image_Pixels contain the output reconstructed data. The Output Valid handshaking signal is used during encoding to indicate if the current bit in Bit Stream is valid. During decoding, the received bit stream is assumed to be compacted by the transmission channel and is always valid and Ready_for_Input is set to 1.

Table 4.2 Data signals for DFS BS SPIHT system

Data signal	Input/Output	Data width
Encode_Decode	Input	1 bit
Image_Pixels	Bidirectional	8 bits
Bit_Stream	Bidirectional	1 bit
Ready_for_Input	Input	1 bit
Output_Valid	Output	1 bit

We then extended the simulations for different image sizes. For these simulations, we used the Alcatel Mietec 0.7μ CMOS technology library [66] with a clock period of 50 ns. Figure 4.16 and Figure 4.17 show the total cell area and clock cycle latency to implement the DFS BS SPIHT system for a resource-driven and timing-driven configuration respectively. The resource-driven configuration is synthesized to minimize cell area by the sharing of common resources. The resource-driven configuration for 32×32 image pixels was synthesized using a total cell area of 157491 gates with a clock cycle latency of 116 cycles. The timing-driven configuration is synthesized to minimize clock cycle latency. The timing-driven configuration for 32×32 image pixels was synthesized using a total cell area of 177965 gates with a clock cycle latency of 65 cycles. For a 13% increase in cell area, the clock cycle latency of the timing-driven configuration is almost half that required by the resource-driven configuration. Further explorations in hardware synthesis can be carried out to obtain a joint optimization between the cell area and clock cycle latency.

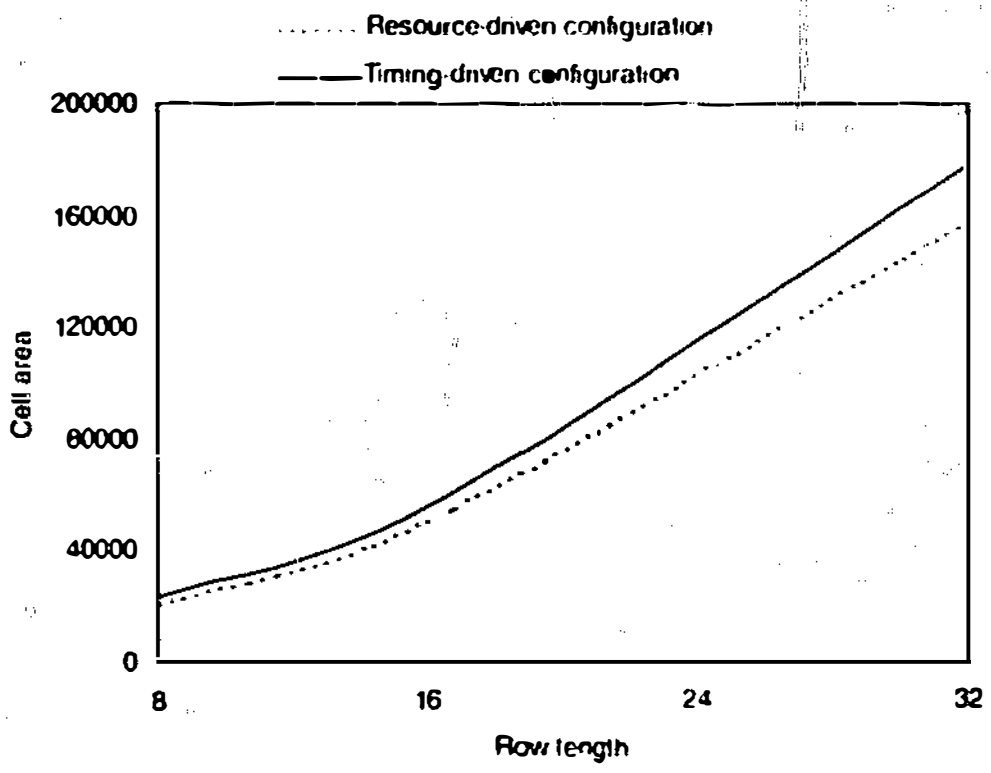


Figure 4.16 Cell area for DFS BS SPIHT configurations

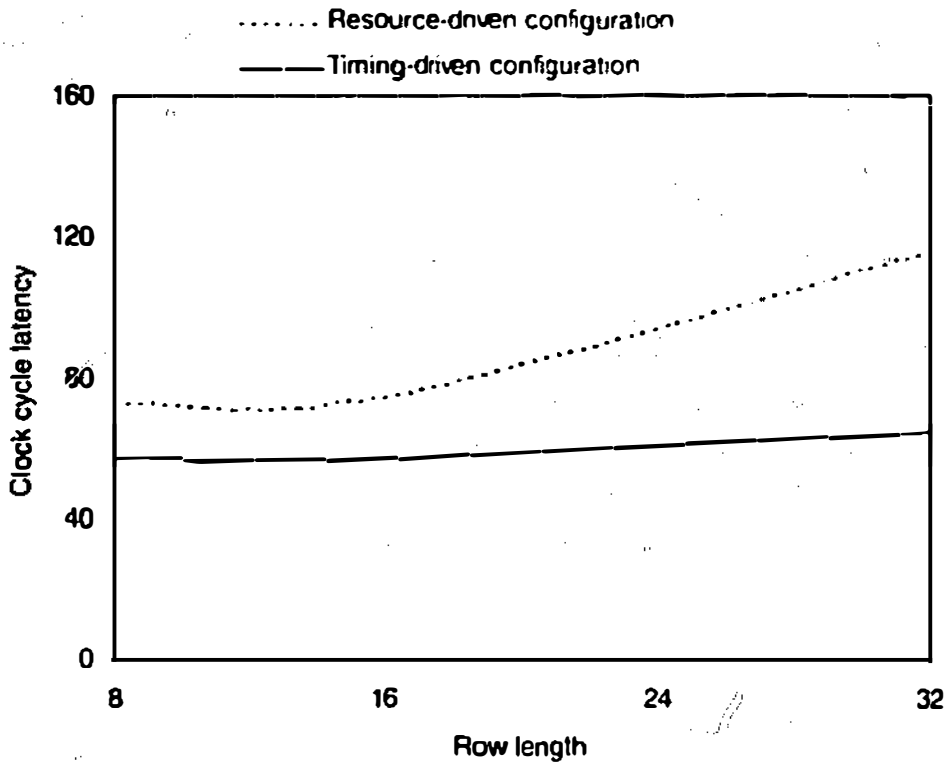


Figure 4.17 Clock cycle latency for DFS BS SPIHT configurations

Figure 4.18 shows the breakdown of the cell area in terms of combinational and sequential areas for the resource-driven configuration. The figure shows that the combinational area increases almost as much as the sequential area as the image size increases. The reason for this is because SYNOPSYS synthesizes memory to static RAM which is implemented using flip-flops and multiplexers in its cell library. A large memory bank would require a correspondingly large multiplexer to access the memory locations. The cell area would be very much reduced by using a RAM component from an IC vendor.

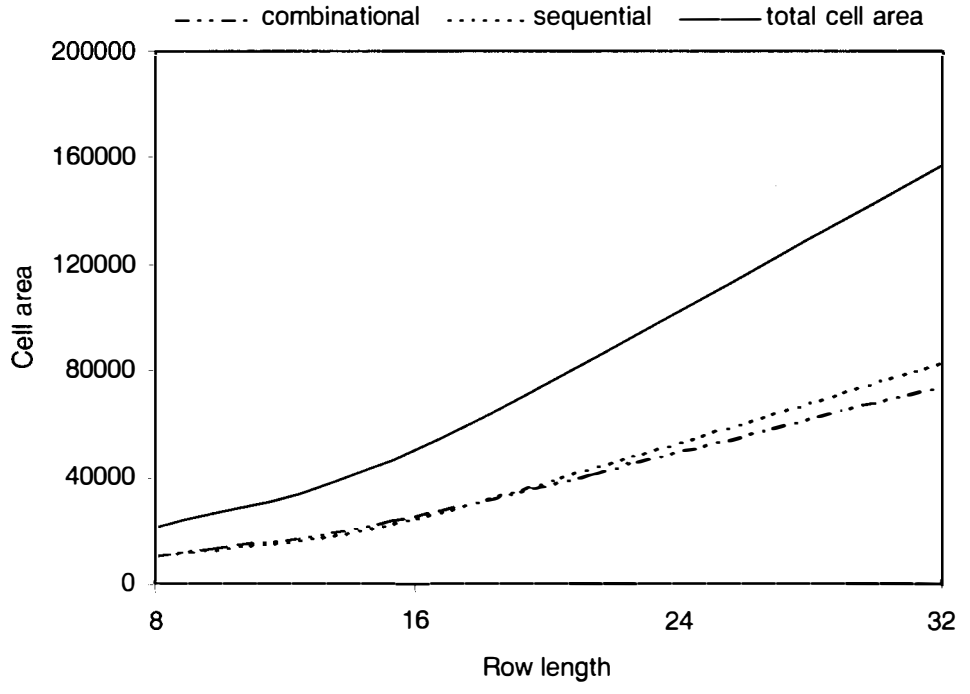


Figure 4.18 Breakdown of cell area showing combinational and sequential areas

Figure 4.19 further shows the total cell area for different bit widths of the coefficient memory bank. The figure shows that the cell area does not increase significantly for larger coefficient bit widths. The increases in the cell areas are due to the additional flip-flops which are required for storing the increased precision in the coefficient magnitudes.

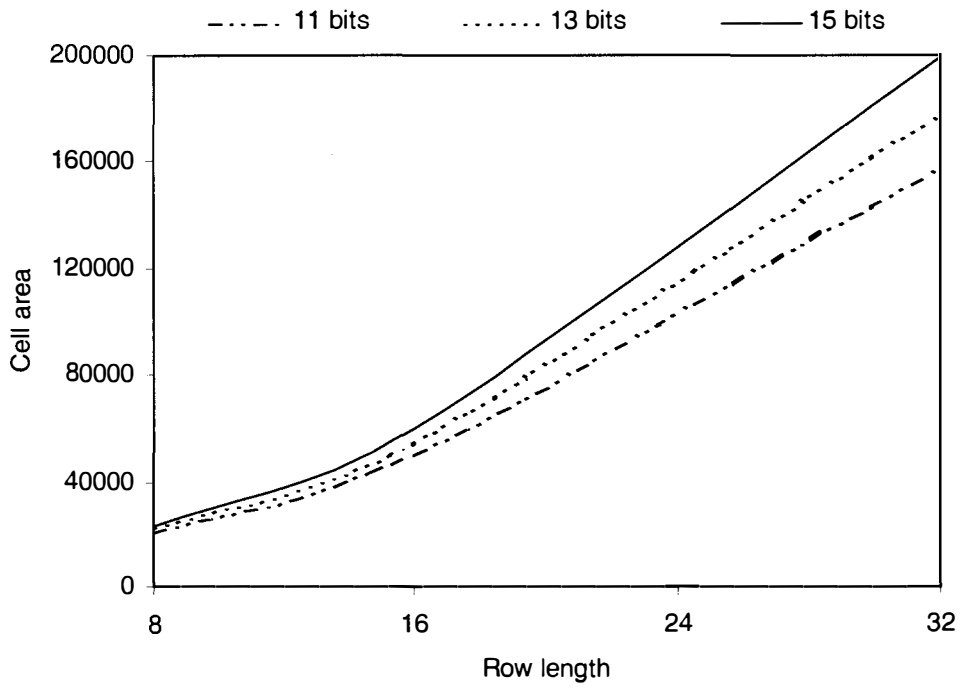


Figure 4.19 Cell area for different widths of coefficient memory bank

4.6 Conclusions

We have presented two approaches for the hardware implementation of the DFS BS SPIHT system. The first approach uses a memory bank approach and the second approach uses a smart pixel (SP) VLSI approach. We have discussed two implementation issues. The first issue is the interfacing of the DWT/IDWT processor with the TS/ITS processor. We have looked at the interfacing of the processors for the memory bank and SP approaches. For the memory bank approach, the DWT coefficient arrangement can be converted to the DFS format by using a look-up table (LUT). The number of entries in the LUT is determined by the size of the subtree. For the SP approach, a different interfacing scheme is used. Due to the nature of the shifting out of the bits in the SP array, the original DFS algorithm is modified so that the DFS is performed on the three orientation branches of the subtree in the First Stage processing.

The second issue is the distribution of the processing and storage requirements amongst the DWT/IDWT and TS/ITS processors to achieve an optimal system. To reduce storage requirements, the coefficient memory bank is used at different stages of processing to store image pixels or wavelet coefficients. One requirement in the system is the generation of the MSIG information. The flexibility of the bit stream approach enables the generation of the MSIG bits before the encoding process. For the SP approach, the MSIG bits can be directly generated inside each pixel. We have also discussed the similarity of processing for the TS encoder and decoder. Both the encoder and decoder use two stages of processing. The First Stage scans from the bottom to the top of the tree whereas the Second Stage scans from the top to the bottom of the tree. Although the First Stage is not essential for decoding, it is advantageous to perform the First Stage scanning to store the MSIG tree structure in the subtree memory bank to simplify the Second Stage decoding. This does not increase the hardware complexity significantly because the memory storage is required for encoding.

Finally, we have presented a memory bank implementation of the DFS BS SPIHT system. The similarity of the encoder and decoder structures enable the construction of a coding system which can be switched to perform encoding and decoding on the same device. The flexibility to perform switching between encoding and decoding makes the coding system very suitable for "encode-decode" applications such as video communicators. The DFS BS SPIHT system can also be applied to "encode-only" applications such as digital cameras and satellite imagers. We have performed simulations to determine the complexity of the system for different configurations, image sizes and coefficient bit widths. The DFS BS SPIHT system is suitable for implementation on DSP processors, FPLDs or VLSI.

Chapter 5

Applications of the DFS Embedded Wavelet Algorithms

5.1 Introduction

Features which are desirable in compression algorithms are high coding performance, the scalability of the coded bit stream, the robustness of the bit stream towards storage or transmission errors, the suitability for content-based coding and rate-control schemes and low complexity for hardware implementation. Embedded wavelet (EW) algorithms give the highest coding performances amongst image coding algorithms in addition to their natural characteristics for bit stream scalability and rate-control [67]. In the previous chapters, we have discussed the DFS EW algorithms which have low complexity for hardware implementation while maintaining the natural advantages of EW coding.

In this chapter, we will look at the applications of the DFS EW algorithms for robust and content-based coding. The DFS algorithms give high coding performance even without arithmetic coding and the reordering of the SAQ. The omissions of arithmetic coding and SAQ reordering in the DFS algorithms were intended to simplify the hardware implementation. An additional benefit of omitting these modules is that the bit stream has robustness towards the occurrence of bit errors. We investigate the robustness of the DFS EZW algorithm. The second application we will look at is for content-based coding for videoconference and videophone applications. In such applications, the normal scene involves a human face occupying the centre of the image and a background. Content-based coding is an important part of the MPEG-4 coding

standard. Each frame of an input video signal is first segmented into a video object plane (VOP). We assume the location of the face VOP and discuss the implementation of the content-based image compression using the DFS implementation of the Improved EZW algorithm.

5.2 Robust Coding

For image or video transmission over a noisy transmission channel, there are two kinds of noise if a lossy compression technique is used to compress the image or video before it is transmitted. The first kind of noise is caused by the compression itself which only encodes the image to an approximated image to be sent through the channel. The other kind of noise is due to the occurrence of errors to the information being transmitted in the noisy channel. Traditionally, these two kinds of noise are handled separately in the source coding and channel coding stages where the coding in each stage is optimized according to the corresponding criteria. Another approach is the joint source and channel coding technique where attempts have been made to obtain an optimum tradeoff between source coding accuracy and channel error protection under the constraint of a fixed transmission bandwidth [68].

During the transmission of the image information in the embedded bit stream using the EZW algorithm, decoding can be stopped at any point and an approximated image can be reconstructed based on the information received so far. This characteristic can also be used in the case that if an error has occurred to the bit stream, then decoding can be terminated and an approximated image can be reconstructed with less than the expected resolution. In order to minimize the propagation of errors, the embedded bit stream can be partitioned into a number of independent bit streams. The partitioned bit

streams can be sent through different transmission channels. An error that occurs in one of the bit streams will not affect the other. The original EZW algorithm will not tolerate any error in any one of the bit streams and it will have to stop decoding the bit stream that has been affected by an error.

An error occurring in a bit position of the EZW encoded bit stream will lead to propagation of errors to other parts of the bit stream. Various methods have been proposed to overcome the propagation of errors. In the work reported in [26], the coefficient tree structure is partitioned into independent subtrees and codes for these subtrees are sent through different channels. In this way, an error occurring in one of the subtrees will not affect the integrity of the other subtrees. In addition, the proposed method can decode the information already received prior to the occurrence of the error to recover as much information as possible. Other methods [27], [29] use similar strategies to isolate the affected part of the codes and decode those not affected.

In this section, we will discuss the error propagation characteristics of each EZW module and investigate different ways to modify the original EZW algorithm including the deletions of some of the modules so as to arrive at modified algorithms that are more robust to channel errors. We propose modifications to the original algorithm that will detect serious transmission errors with additional information and will ignore transmission errors that only cause local effects.

5.2.1 Error Characteristics of the DFS EZW Algorithm

With the arithmetic coder (AC) in the EZW system, an error occurring in any part of the bit stream will force the AC in the decoder side of the system to go out of synchronization with the AC in the encoder. The EZW algorithm performs a raster scan

of the coefficients to establish the ancestor-descendant relationship in the coefficient tree structure. In each pass of the scan, this relationship is represented in the MAP bit stream. An error occurring in a bit position may corrupt the information about the rest of the tree. From that point onwards, the decoder will not be able to decode any further useful information. The decoder is still able to detect the occurrence of an error by the AC [26]. It may still use information already received to construct a decoded image of less resolution. In the following discussion, we assume that the arithmetic coder is optional so as to study the effects of an error on the bit stream being transmitted.

Using the DFS representation, the coefficient tree structure is naturally partitioned into a number of independent subtrees. These subtrees are encoded separately during transmission. An error occurring during the transmission of the encoded information in a pass will only affect the information in a subtree if different physical or logical channels are used to transmit the information of the subtrees. Similarly, the decoder will be able to use already received information to construct image information from the part of the subtree. Normally, an error occurring in a bit position in the SAQ bit stream will only affect the corresponding coefficient, i.e. the value of the coefficients may be increased or decreased. In the EZW system, although an error occurring in SAQ will not affect the decoded structure of the coefficient tree, it will affect more than one coefficient because of the SAQ reordering. An error to an approximate bit will change the value of that coefficient. While the encoder is still using the correct value to order the approximated bits, the decoder will use the erroneous bit to order the decoded coefficient values in the decoder. As a result, approximated values will be added to the wrong coefficients.

5.2.2 Robust DFS EZW Models

The underlying modification made to the EZW algorithm to improve its robustness is to use the DFS to partition the coefficient tree into independent subtrees where the MAP and SAQ bits can be sent through different physical or logical channels. If arithmetic coding is performed on the overall bit stream containing both MAP and SAQ, decoding will stop at or close to the error point for this subtree. However, this error will not affect the other subtrees. If arithmetic coding is applied only to the MAP and no reordering is done on the SAQ, then an error occurring in the SAQ part of the bit stream will only affect the corresponding coefficient. In this case, the bit errors can be tolerated and decoding does not need to be stopped.

On the other hand, if only the SAQ is arithmetically coded, an error occurring in either part of the bit stream may lead to termination of decoding. However, we will discuss the situation that an error to the MAP part may be tolerated if there is no structural change to the tree structure. If no arithmetic coding is used, then decoding can continue if the error does not lead to the structural change of the MAP. Table 5.1 summarizes all the cases where an error to a symbol in the MAP changes the symbol to another symbol and their effects. Of the nine cases, the change of a POS symbol to a NEG symbol and vice versa will not be detected without the AC. The effect of this error is to change the sign of the coefficient in the decoder and this change of sign will lead to noise being introduced into the decoded image. Errors leading to the other cases can be detected if the length of the MAP is known. Assuming the length of the MAP is known to the decoder, an error causing another symbol to change to the ZTR symbol and vice versa would be detected within the current MAP. A more serious case, as far as error detection is concerned, is that an error causes the change of the symbols POS or

NEG to IZ and vice versa. In this case, the error will not be detected until the decoding of the next MAP. In order to detect this error effectively, the decoder also needs to know the length of the SAQ. Table 5.2 summarizes the detection of errors and the severity if undetected. The use of the AC to detect any error is not listed in the table except when the error can only be detected by it.

Table 5.1 Change of a MAP symbol to another symbol by a one-bit error

MAP symbol	Corrupted symbol	Effect to the decoder
POS	NEG	The sign of one coefficient is changed to negative
	IZ	The corresponding SAQ is one bit shorter
	ZTR	The length of MAP is different
NEG	POS	The sign of one coefficient is changed to positive
	IZ	The corresponding SAQ is one bit shorter
	ZTR	The length of MAP is different
IZ	POS	The corresponding SAQ becomes a bit longer
	NEG	The corresponding SAQ becomes a bit longer
	ZTR	The length of MAP is different
ZTR	POS	The length of MAP is different
	NEG	
	IZ	

Table 5.2 Error detection of the DFS EZW

An error to	Detected by	Severity if undetected
Reordered SAQ	Arithmetic coder only	Most coefficients in a subtree
POS ↔ NEG		The sign of one coefficient
Non-reordered SAQ		The magnitude of one coefficient
MAP symbol involving IZ	Decoder even without AC, but with the lengths of MAP and SAQ	Changing the lengths of SAQs in subsequent passes and information on one significant coefficient
MAP symbol involving ZTR		Major corruption to the structure of the tree and information on significant coefficients

From Table 5.2, if the approximated bits in the SAQ are reordered, then an AC could be used to detect any error occurring to it. If arithmetic coding is not used, the SAQ should not be reordered. The most destructive cases involve the symbol ZTR whether it is the original symbol or the corrupted symbol. The net effect of this error is that the information following this error becomes almost random. In this case, decoding of the corresponding subtree should be stopped. The less destructive cases involve the IZ symbol. In these cases, the structure of the MAP is not changed, except that the length of SAQ is one bit different. If the decoder can identify that such an error has occurred, it can still decode the MAPs in subsequent passes but it has to discard all the future SAQs. Decoding a MAP can only extract the most significant bits of the newly found significant coefficients.

5.2.3 Simulation Results

In this section, we perform simulations to investigate the robustness and tolerance of the DFS EZW algorithm to channel noise. In the simulations, we did not use the AC and SAQ reordering so as to test the tolerance of the algorithm to the POS $\leftrightarrow$ NEG errors and the occurrence of errors to the non-reordered SAQ, i.e. these errors are not detected. The coefficient tree structure was partitioned into independent subtrees using the DFS representation. Each subtree is coded independently as in [26] and the embedded bit stream for each subtree is assumed to be sent through separate physical or virtual transmission channels. Table 5.3 shows the PSNR obtained when one random bit error was substituted for each 1000 SAQ symbols for different bit rates (bpp). The table shows the total number of SAQ symbols and the total number of SAQ bit errors at different bit rates. As expected from Table 5.2, since each SAQ bit error affects only

one coefficient, the PSNR is not much affected by the SAQ bit errors. Figure 5.1 shows the Lena image at 1.0 bpp with 23 SAQ bit errors. A PSNR of 34.39 dB was obtained.

Table 5.3 Bit errors in SAQ symbols without error detection

bpp	Number of SAQ symbols	Number of SAQ bit errors	PSNR (dB)	
			without bit error	with bit error
0.25	5511	4	28.25	28.24
0.5	11227	10	30.89	30.88
1.0	23493	23	34.39	34.39

Figure 5.1 Lena image at 1.0 bpp with 23 SAQ bit errors: PSNR = 34.39 dB

Similarly, Table 5.4 shows the PSNR obtained when one random bit error was substituted for each 1000 POS ↔ NEG symbols for different bit rates. The random bit error changed the symbol POS to the symbol NEG and vice versa. The table shows the total number of POS/NEG symbols and the total number of POS ↔ NEG bit errors at

different bit rates. A change of the POS symbol to the NEG symbol and vice versa will lead to the sign of the decoded coefficient being changed and as a consequence noise will be introduced into the decoded image. However, the results show that at higher bit rates, the noise in the image due to each POS ↔ NEG bit error is much reduced. Figure 5.2 shows the Lena image at 1.0 bpp with 22 POS ↔ NEG bit errors. A PSNR of 34.26 dB was obtained.

Table 5.4 Bit errors in POS ↔ NEG symbols without error detection

bpp	Number of POS/NEG symbols	Number of POS ↔ NEG errors	PSNR (dB)	
			without bit error	with bit error
0.25	4154	3	28.25	28.18
0.5	9727	10	30.89	30.61
1.0	24027	22	34.39	34.26

Figure 5.2 Lena image at 1.0 bpp with 22 POS ↔ NEG bit errors: PSNR = 34.26 dB

5.3 Content-Based Coding

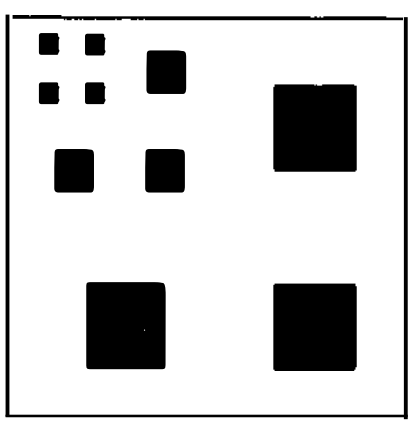
In applications where the purpose of the image or video is to provide face-to-face interaction at different ends of the communication channel, more emphasis should be placed on the human face in the image and less on the background. In videoconference and videophone applications, the normal scene involves a human face occupying the centre of the image and a background. While the background serves the purpose of showing the context of the overall picture, its details are less important and viewers are normally less interested in its contents. A complete elimination of the background may not be acceptable. Methods have been proposed to allocate more resources to the region of an image which contains the human face in interest [69], [70]. The semethods are based on analyzing the given image and on locating the face region.

In this section, we propose to use the EZW algorithm to accommodate content-based encoding of different regions of an image. Assuming the information about where the region of interest in an image is available, the proposed method applies a scaling factor to the coefficients in the selected region in all the subbands of the resulting wavelet decomposition. The coefficients are then encoded and transmitted to the other end of the communication channel. Before the decoded approximated coefficients are inverse wavelet transformed, they are multiplied by the inverse of the corresponding scaling factor. The advantages of the proposed method are that the encoded bit stream does not lose its embedded and scalable features and there will not be serious blocking artifacts at the boundaries of the scaled up and the background regions. The proposed method can be extended to video coding schemes which are based on the EZW algorithm.

5.3.1 Content-Based EZW Encoding

Using the EZW algorithm, to place more emphasis on the region of interest is to increase the significance of the coefficients in that region. The most straightforward way to increase the significance of the coefficients is to multiply those coefficients with a scaling factor before the coefficients are subjected to the scanning by the EZW algorithm. In Figure 5.3(a), once the region is determined in the input image, the region is then converted into the corresponding regions in all the subbands. Figure 5.3(b) shows the selected region of the image in all the subbands for a three scale wavelet decomposition of an image.

(a) Selected region



(b) Selected region in the subbands

Figure 5.3 Selected region of an image and its position in the subbands

Figure 5.4 shows the steps involved in the content-based image encoding and decoding scheme. The first step of the scheme is the location of the region of interest in the input image. This process can be manual or automated depending on the application. The location and size of the face region are then transmitted. In the case of video compression, this information may not need to be transmitted for every frame once the region is located and is tracked by both the encoder and decoder.

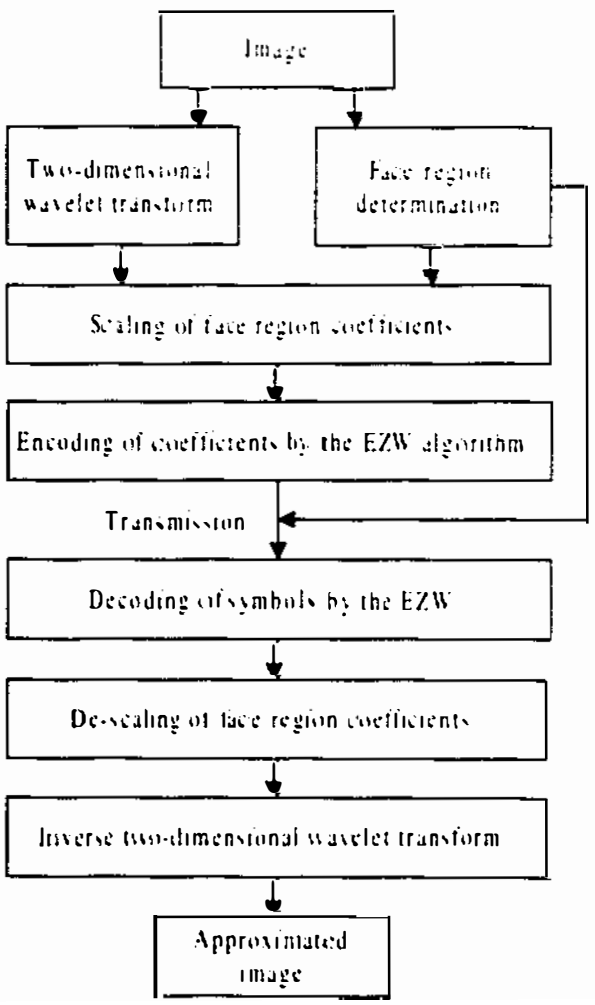


Figure 5.4 Content-based encoding and decoding of image using the EZW algorithm

The image is then subjected to a two-dimensional wavelet decomposition. After the coefficients in the selected region are multiplied by the scaling factor, they are processed by the EZW algorithm as if they were normal coefficients from a wavelet

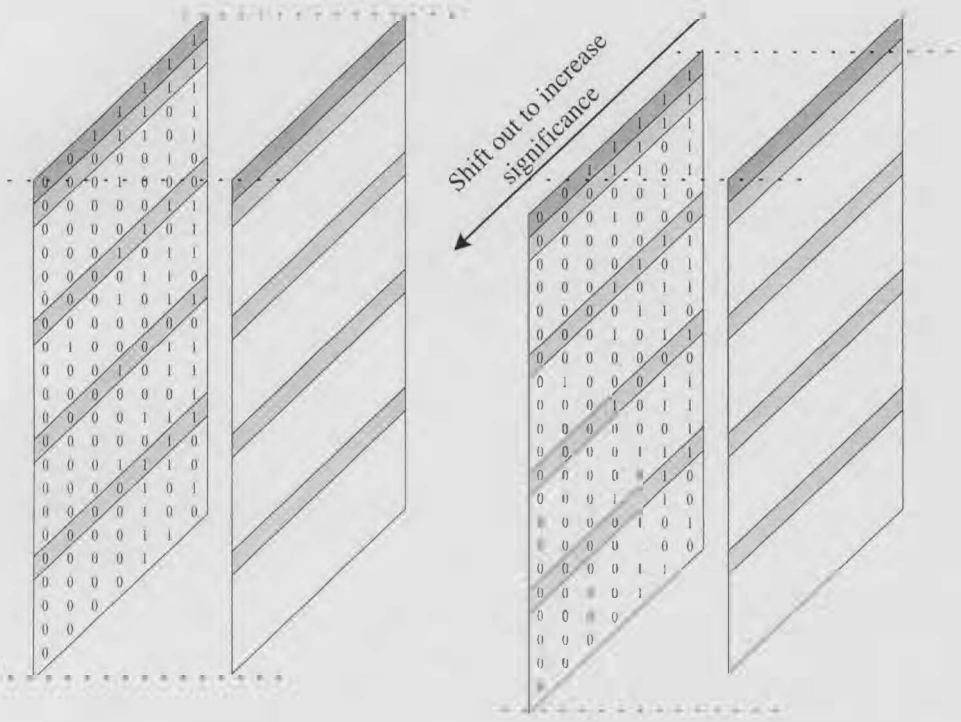
decomposition. There is no change to the EZW algorithm itself and the algorithm produces the EZW symbols according to the modified values of the coefficients. At the decoder end, the EZW symbols are decoded. Before the decoded values of the coefficients are inverse transformed back into the spatial domain, the coefficients in the selected region of the image in all the subbands are multiplied by the inverse of the scaling factor.

In the scheme shown in Figure 5.4, due to the fact that there has been an artificial increase in the significance of the coefficients in the selected region, there may be an increase in the number of ZTR symbols in the first few passes of the EZW algorithm. The number of these introduced ZTR symbols depends on the size of the scaling factor. Let the maximum value of the magnitudes of the coefficients in the non-selected region be C_{max}, the maximum value of the magnitudes of the original coefficients in the selected region be C'_{max} and the scaling factor be S. After the coefficients in the selected region are multiplied by the scaling factor S, the maximum value is $S * C'_{max}$. If $S * C'_{max} > C_{max}$, then the threshold value of the EZW algorithm is equal to $S * C'_{max} / 2^N$ where N is the N^{th} pass of the coefficients. Let $T = S * C'_{max} / 2^N$. During the N^{th} pass, if $T > C_{max}$, then all the subtrees belonging to the non-selected region have no significant coefficients and each of these subtrees is encoded by a ZTR symbol. In general, the number of ZTR symbols is increased if $S * C'_{max} > C_{max}$.

5.3.2 DFS BS EZW Content-Based Implementation

Figure 5.5(a) shows a three-dimensional representation of the wavelet coefficients in sign-magnitude binary format. The sign bits of the coefficients are processed separately according to the EZW algorithm. In this binary representation of the magnitudes of the

coefficients, each bit plane of the subtrees is scanned in successive passes according to the bit stream (BS) version of the DFS EZW algorithm starting from the most significant bit plane. If the scaling factor is of the power of two, the multiplying of the selected area in all the subbands by the scaling factor is equivalent to shifting the subtrees forward by the log of the scaling factor.



(a) Subtree bit planes (b) Shifted bit planes

Figure 5.5 Three-dimensional representation of sign-magnitude binary format

Shown in Figure 5.5(b) is the case where the subtrees for a selected region of the image have been shifted forward. Scanning of the subtrees starts from the outmost bit plane where only the bits for the selected region are present. This arrangement avoids the generation of the ZTR symbols as discussed in Section 5.3.1 in the case of the general EZW algorithm. The other feature of the DFS BS EZW implementation of content-based image encoding is that the shifting forward of the subtrees can be performed even after the scanning of the subtrees has started. This occurs in

applications where the user at the decoder end of the transmission may request more information for a selected region to be transmitted. Using the DFS BS EZW implementation, the request is translated to just shifting forward of those subtrees corresponding to the selected region in a new pass of the next bit plane.

5.3.3 Simulation Results

In this section, the DFS BS implementation of the Improved EZW algorithm is used in simulating the performance of the content-based encoding of an image on the 512×512 Lena image. Using the face region selected by a rectangle as shown in Figure 5.3(a), Figure 5.6 shows the resulting images using scaling factors as shown in Table 5.5 together with the overall PSNRs. Decoding of the image stopped at the bit rate of 0.1 bpp. As seen from the figures, the decoded image without content-based encoding loses sharpness more or less uniformly throughout the image.

Compared to the face region in those decoded images with scaling factors greater than one, the sharpness of the face region improves with increasing scaling factor, while the sharpness of the background deteriorates, as well as the overall PSNRs. Although an abrupt window is used to select the face region, there is no obvious blocking effect at the boundary between the selected region and the non-selected region. A 'blocking' effect becomes obvious when the selected region is much clearer than the non-selected region.

Table 5.5 Overall PSNRs of content-based encoded Lena images at 0.1 bpp

Scaling factor	1	4	16
PSNR (dB)	28.48	26.55	23.51

(a) Scaling factor is 1 at 0.1 bpp

(b) Scaling factor is 4 at 0.1 bpp

(c) Scaling factor is 16 at 0.1 bpp

Figure 5.6 Content-based decoded Lena images at 0.1 bpp

To show the performance of the proposed method at very low bit rates, decoding of the image stopped at 0.05 bpp. Figure 5.7 shows the resulting images and Table 5.6 the corresponding PSNRs versus the scaling factor. Without content-based encoding, every part of the image is equally affected by the low bit rates used, especially the face region of the image as shown in Figure 5.7(a). The sharpness of the face region is improved by using a scaling factor greater than one as shown in Figure 5.7(b) and Figure 5.7(c) with scaling factors equal to 4 and 16 respectively. The face region is even sharper in Figure 5.7(c) than that in Figure 5.6(a) where there is no content-based encoding with double the bit rate. From these simulations, a scaling factor of 16 increases the contrast of sharpness between the selected region and the background where it becomes almost unrecognizable.

Table 5.6 Overall PSNRs of content-based encoded Lena images at 0.05 bpp

Scaling factor	1	4	16
PSNR (dB)	26.32	24.78	21.39

(a) Scaling factor is 1 at 0.05 bpp

(b) Scaling factor is 4 at 0.05 bpp

(c) Scaling factor is 16 at 0.05 bpp

Figure 5.7 Content-based decoded Lena images at 0.05 bpp

5.4 Conclusions

In this chapter, we have discussed two applications for the DFS EW algorithms. The first application we have looked at is for robust coding. We have looked into the error characteristics of the EZW algorithm. The original EZW algorithm using arithmetic coding can be used to detect all the errors to the transmitted bit stream. Decoding will stop and information already received by the decoder can be used to reconstruct an approximated image with less resolution. By partitioning the coefficient tree structure into independent subtrees using the DFS representation, the effect of an error occurring in a subtree is confined to that subtree only. The original EZW algorithm does not tolerate any error in the bit stream because of the AC. We have listed three other different cases to apply the AC to part of the bit stream. Each of the cases offer different error detection capability and error tolerance.

We have also looked into the severity of error occurrence to different parts of the bit stream and have found that the reordering of the approximate bits in SAQ will lead to more error propagation if not detected. For the MAP, we have analyzed the different cases when an error occurs to each of the four MAP symbols. It has been suggested that decoding does not need to be stopped if the error does not lead to structural change to the transmitted information about the subtrees. We have found that the effects of bit errors range from affecting only a fraction of the magnitude of a coefficient, the sign of a coefficient, the length of the SAQ only, and the most serious case, when both the lengths of the MAP and SAQ are affected.

Simulations have been performed to look at the actual behaviours of the various modifications to the EZW algorithm. The first modification was the DFS partitioning with no AC and no SAQ reordering. This modified configuration has two characteristics. It detects the major errors that will lead to error propagation and tolerates those errors that will only cause localised errors. However, without the AC, we need to assume that the lengths of both the MAP and SAQ will be available to the decoder and they are not subject to errors. Further analysis is required to decide if this information should be incorporated into the source coding or be provided by the channel coding. The use of this additional information on the lengths of MAP and SAQ may enable us to identify the errors involving the IZ and ZTR symbols.

The second application we have looked at is for content-based coding. We have presented a new method to implement the approach where a selected region of an image is to be encoded with more resources. The method is based on the DFS representation of the EZW algorithm and makes use of the bit stream implementation. The approach is modular in the sense that the additional feature of content-based encoding and decoding

is performed by a separate function while the modules of the original DFS BS EZW structure is unchanged. The DFS BS subtree representation of the wavelet coefficients also makes it possible for implementing scaling factors which are powers of two by simply shifting the subtrees to increase the significance of the corresponding coefficients at the encoder. At the decoder, the subtrees are shifted in the reverse direction before the coefficients are inverse wavelet transformed. This simple shifting of the subtrees is useful for interactive selection of regions in an image to be given more resources even when some of the bit planes have been transmitted. The other advantage of the proposed method is its adaptability in that the sharpness of the background as well as the selected region can be improved with higher bit rates which can be determined during encoding or decoding.

Simulation results have confirmed that the proposed method gives a much clearer image in the selected region at the expense of losing sharpness in the background region as well as a drop in the overall PSNR. However, the method does not produce any obvious blocking artifacts apart from an increase in contrast of the sharpness between the selected region and the background with a large scaling factor. The contrast in sharpness can be improved by using multiple regions around the target region with progressively decreasing scaling factors.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We began this thesis by asking the question whether a single coding algorithm can be designed to meet the diverse requirements for image communication over wireless transmission channels. Embedded wavelet (EW) algorithms show much potential to meet the requirements of high coding performance, bit stream scalability and robustness to transmission errors. Other desirable features are low complexity architectures and suitability for content-based coding schemes. The barrier impeding the potential of EW algorithms are tree searching (TS) schemes which can be efficiently implemented in hardware. There are two approaches for TS architectures: iterative or noniterative. The noniterative architectures suffer from misprediction. The complexity in the implementation of iterative TS architectures is that it requires keeping track of the significance of the descendant nodes.

In this work, we focused on iterative TS schemes and used a top-down design methodology beginning from the investigation of the original EW algorithms. We looked at variations in EW algorithms and identified four variations in which the algorithms differ from one another. The algorithms differ in the tree structure, the search strategy, the coding scheme and the output scheme. One search strategy is the depth-first search (DFS). The DFS gives several advantages for hardware implementation. The DFS algorithms have low storage requirements, simple computations and allows for efficient addressing of the nodes in the tree structure. We

performed simulations to compare the DFS variants with its original algorithms and found that the coding performance is only slightly lower than the original algorithms. Furthermore, the performance of the DFS SPIHT variant without arithmetic coding is almost comparable to the performance of the complete EZW algorithm with arithmetic coding. A comparison with the WVQ algorithm [33] shows that the performance of the DFS SPIHT algorithm is higher than the WVQ algorithm at high bit rates and is comparable to the WVQ algorithm at lower bit rates.

At the architecture level, the DFS allows for novel bit stream (BS) TS architectures where the magnitude bits are not stored and are processed as they flow through the architecture resulting in fast architectures with minimal storage and low complexity. Two stages are required in the processing. In the First Stage, the coefficient tree is scanned from the bottom to the top of the tree and the significance information is propagated from the leaves to the roots. The Second Stage scans the tree from the top to the bottom of the tree and performs the coding. The DFS performs a natural partitioning of the coefficient tree into subtrees which can be processed in parallel for faster processing or to meet a target bit rate. Parallel processing can also be used to lower the clock frequency to reduce the overall power consumption of the architecture. Another advantage of the parallel DFS architecture is that it provides options to output in different formats to increase coding performance.

The DWT/IDWT and TS/ITS processors can be combined to form a single architectural structure for implementation. The processing and storage requirements are distributed amongst the processors to achieve an optimal system. The similarity of the encoder and decoder structures enable the construction of a coding system which can be switched to perform encoding and decoding on the same device. We have performed

simulations to determine the complexity of the system for different configurations, image sizes and coefficient bit widths. The DFS BS SPIHT system is suitable for implementation on DSP processors, field programmable logic devices or VLSI and is a promising solution for real-time video coding applications in general and for portable video communicators in particular.

We have looked at two applications for the DFS EW algorithms. The DFS algorithms have robustness to transmission errors by not performing SAQ reordering and arithmetic coding. The DFS algorithms can accommodate content-based coding by shifting the subtree bitplanes before the encoding process.

6.2 Future Work

The work presented in this thesis can be extended in the following ways. Firstly, the DFS algorithms and architectures can be extended to accommodate motion compensation schemes for coding of colour video images [71] using newer biorthogonal wavelet filters [72]. We have performed initial work on applying a winner-take-all artificial neural network for motion compensation schemes for the DFS algorithms [9]. Initial simulations show the potential of this approach. Secondly, the DFS EW algorithms with its corresponding bit stream architectures form a good foundation for video communicators. The robustness and content-based coding schemes could form the front-end and the back-end of the DFS EW system as shown in Figure 6.1. The front-end module performs the image segmentation for the system. The back-end error protection module protects the MAP bits for transmission. The SAQ bits do not have to be protected for transmission.

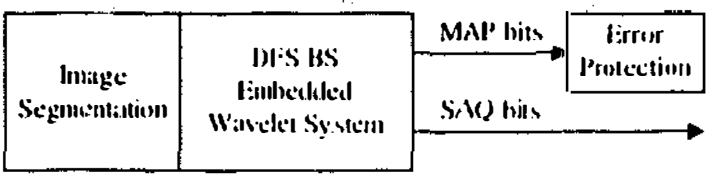


Figure 6.1 DFS BS EW system with image segmentation and error protection modules

Thirdly, the DFS traversal scheme can be extended to accommodate the searching of non-regular tree structures such as those resulting from decomposition using the discrete wavelet packet transform. An EW algorithm which uses non-regular tree structures is the Space-Frequency Quantization (SFQ) algorithm [18]. Fourthly, the work in this thesis can be improved using techniques used in the Embedded Block Coding with Optimized Truncation (EBCOT) algorithm [73] which is used in the JPEG2000 standard. The EBCOT algorithm adds the feature of spatial scalability to EW algorithms.

Bibliography

[1] K. Eshraghian, S. Lachowicz, G. Alagoda and L. Ang, "Architectural mappings for multimedia smart-pixel arrays," in *Proc. IEEE Int. Workshop Design, Test and Applications*, Dubrovnik, Croatia, pp. 33-35, Jun. 1998.

[2] L. Ang, H. N. Cheung and K. Eshraghian, "VLSI architecture for significance map coding of embedded zerotree wavelet coefficients," in *Proc. IEEE Asia Pacific Conf. Circuits and Systems*, Chiangmai, Thailand, pp. 627-630, Nov. 1998.

[3] L. Ang, H. N. Cheung and K. Eshraghian, "VLSI architecture for embedded zerotree wavelet coding," in *Proc. 5th Int. Symp. DSP for Communication Systems*, Perth, Australia, pp. 128-133, Feb. 1999.

[4] L. Ang, H. N. Cheung and K. Eshraghian, "VLSI decoder architecture for embedded zerotree wavelet algorithm," in *Proc. IEEE Int. Symp. Circuits and Systems*, Orlando, U.S.A, pp. 141-144, May 1999.

[5] L. Ang, H. N. Cheung and K. Eshraghian, "EZW algorithm using depth-first representation of the wavelet zerotree," in *Proc. 5th Int. Symp. Signal Processing and its Applications*, Brisbane, Australia, pp. 75-78, Aug. 1999.

[6] L. Ang, H. N. Cheung and K. Eshraghian, "Robust image compression using the depth-first search on the wavelet zerotree," in *Proc. 5th Int. Symp. Signal Processing and its Applications*, Brisbane, Australia, pp. 797-800, Aug. 1999.

[7] H. N. Cheung, G. Alagoda, K. Eshraghian and L. Ang, "Smart-pixel VLSI architecture for embedded zerotree wavelet coding," in *Proc. 5th Int. Symp. Signal Processing and its Applications*, Brisbane, Australia, pp. 693-696, Aug. 1999.

[8] L. Ang, H. N. Cheung and K. Eshraghian, "VLSI architecture for very high resolution scalable video coding using the virtual zerotree," in *Proc. IEEE Workshop Signal Processing Systems*, Taipei, Taiwan, Oct. 1999.

[9] L. Ang, H. N. Cheung and K. Eshraghian, "Comparison of winner-take-all motion compensation schemes for embedded wavelet coding," in *Proc. 6th Int. Conf. Neural Information Processing*, Perth, Australia, pp. 390-394, Nov. 1999.

[10] H. N. Cheung, L. Ang and G. Alagoda, "Bit stream architecture for the implementation of the improved embedded zerotree wavelet algorithm," in *Proc. Second Int. Conf. Information, Communications & Signal Processing*, Singapore, Dec. 1999.

[11] H. N. Cheung and L. Ang, "Analysis of embedded zerotree wavelet algorithms for robust image compression," in *Proc. Second Int. Conf. Information, Communications & Signal Processing*, Singapore, Dec. 1999.

[12] H. N. Cheung, L. Ang and K. Eshraghian, "Embedded zerotree wavelet processor for mobile video communicator," in *Proc. IEEE Int. Symp. Intelligent Signal Processing and Communication Systems*, Phuket, Thailand, Dec. 1999.

[13] H. N. Cheung, L. Ang and K. Eshraghian, "Parallel architecture for the implementation of the embedded zerotree wavelet algorithm." in *Proc. 5th Australasian Computer Architecture Conf.*, Canberra, Australia, Jan. 2000.

[14] L. Ang, H. N. Cheung and K. Eshraghian, "A dataflow-oriented VLSI architecture for a modified SPIHT algorithm using depth-first search bit stream processing," in *Proc. IEEE Int. Symp. Circuits and Systems*, Geneva, Switzerland, pp. 291-294, May 2000.

[15] H. N. Cheung and L. Ang, "Application of the EZW algorithm to content-based image compression," to be presented at *IEEE Int. Symp. Intelligent Signal Processing and Communication Systems*, Honolulu, Hawaii, Nov. 2000.

[16] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing*, vol. 41, pp. 3445-3462, Dec. 1993.

[17] A. Said amd W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 6, pp. 243-250, Jun. 1996.

[18] K. Ramchandran, Z. Xiong and M. T. Orchard, "Space-frequency quantization for wavelet image coding," *IEEE Trans. Image Processing*, vol. 6, no. 5, pp. 677-693, May 1997.

[19] J. M. Zhong, C.H.Leung and Y.Y.Tang, "An improved embedded zerotree wavelet image compression algorithm based on significance checking in wavelet trees," in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics*, pp. 4567-4571, Oct. 1998.

[20] S. A. Martucci, I. Sodagar, T. Chiang and Y. Zhang, "A zerotree wavelet video coder," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, pp. 109-118, Feb. 1997.

[21] Q. Wang and M. Ghanbari, "Scalable coding of very high resolution video using the virtual zerotree," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, pp. 719-727, Oct. 1997.

[22] I. Sodagar, H. -J. Lee, P. Hatrack and B. -B. Chai, "Multi-scale zerotree entropy coding," in *Proc. IEEE Int. Symp. Circuits and Systems*, Geneva, Switzerland, pp. 311-314, May 2000.

[23] P. N. Topiwala (Ed.), *Wavelet image and video compression*, Kluwer Academic Publishers, Jul. 1998.

[24] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, Oct. 1996.

[25] M. Vetterli, *Wavelets and subband coding*, Prentice Hall, Apr. 1995.

[26] C. D. Creusere, "A new method of robust image compression based on the embedded zerotree wavelet algorithm," *IEEE Trans. Image Processing*, vol. 6, pp. 1436-1442, Oct. 1997.

[27] H. Man, F. Kossentini and M. J. T. Smith, "A class of EZW image coders for noisy channels," in *Proc. Int. Conf. Image Processing*, vol. 3, pp. 90-93, Oct. 1997.

[28] H. Man, F. Kossentini, and M. J. Smith, "A channel error robust EZW image coding technique," in *Proc. 2nd Erlagen Symposium on Advances in Digital Image Communication*. Apr. 1997.

[29] P. G. Sherwood and K. Zeger. "Progressive image coding for noisy channels," *IEEE Signal Processing Letters*. vol. 4, pp. 189-191, Jul. 1997.

[30] P. G. Sherwood and K. Zeger. "Progressive image coding on noisy channels," in *Proc. Data Compression Conf.*. pp. 72-81. Mar. 1997.

[31] T. -C. Yang, S. Kumar and C. -C. J. Kuo, "Low-overhead error-resilient bit-plane image coding," in *Proc. IEEE Int. Symp. Circuits and Systems*, Orlando, U.S.A, pp. 50-53, May 1999.

[32] Y. Q. Shi and H. Sun, *Image and video compression for multimedia engineering*, CRC Press, Dec. 1999.

[33] S. -K. Paek and L. -S. Kim, "A real-time wavelet vector quantization algorithm and its VLSI architecture," *IEEE Trans. Circuits and Systems for Video Technology*. vol. 10, no. 3, pp. 475-489, Apr. 2000.

[34] J. Bae and V. K. Prasanna, "A fast and area-efficient VLSI architecture for embedded image coding," in *Proc. Int. Conf. Image Processing*, vol. 3, pp. 452-455, Oct. 1995.

[35] M. Schwarzenberg, M. Traeber, M. Scholles and R. Schuffny, "A VLSI chip for wavelet image compression," in *Proc. IEEE Int. Symp. Circuits and Systems*, Orlando, pp. 271-274, May 1999.

[36] R. Y. Omaki, G. Fujita, T. Onoye and I. Shirakawa, "Architecture of embedded zerotree wavelet based real-time video coder," in *Proc. IEEE Int. Conf. ASIC/SOC*, pp. 137-141, Sep. 1999.

[37] J. Singh, A. Antoniou and D. J. Shpak, "Hardware implementation of a wavelet based image compression coder," in *Proc. IEEE Symp. Advances in Digital Filtering and Signal Processing*, pp. 169-173, Jun. 1998.

[38] Vega-Pineda, J., Suriano, M. A., Villalva, V. M., Cabrera, S. D. and Chang, Y. – C., "A VLSI array processor with embedded scalability for hierarchical image compression," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 4, pp. 168-171, May 1996.

[39] V. Bhaskaran and K. Konstantinides, "Image and video compression standards : algorithms and architectures," Kluwer Academic Publishers, Jun. 1997.

[40] W.-K. Lin and N. Burgess, "Listless zerotree coding for color images", in *Proc. 32nd Asilomar Conf. Signals, System and Computers*, CA, USA, Nov. 1998.

[41] W.-K. Lin and N. Burgess, "Low memory color image zerotree coding", in *Proc. Information, Decision and Control*, pp. 91-95, Adelaide, Australia, Feb. 1999.

[42] W.-K. Lin, B. W.-H. Ng, N. Burgess and A. Bouzerdoum, "Reduced memory zerotree coding algorithm for hardware implementation", in *Proc. IEEE Int. Conf. Multimedia Computing and System*, Florence, Italy, Jun. 1999.

[43] C.-Y. Su and B.-F. Wu, "Image coding based on embedded recursive zerotree", in *Proc. Int. Sym. Multimedia Information Processing*, Taipei, Taiwan, Dec. 1997.

[44] W.-K. Lin and N. Burgess, "A low memory video compression algorithm for hardware implementation", in *Proc. Int. Workshop Multimedia Signal Processing*, Copenhagen, Denmark, Sep. 1999.

[45] W.-K. Lin and N. Burgess, "3D listless zerotree coding for low bit rate video", in *Proc. Int. Conf. Image Processing*, Kobe, Japan, Oct. 1999.

[46] E. H. Adelson, E. P. Simoncelli and R. Hingorani, "Orthogonal pyramid transforms for image coding," in *Proc. SPIE*, vol. 845, Oct. 1987, pp. 50-58.

[47] I. H. Witten, R. Neal and J. G. Cleary, "Arithmetic coding for data compression," *Comm. ACM*, vol. 30, pp. 520-540, Jun. 1987.

[48] V. R. Algazi and R. R. Estes, "Analysis based coding of image transform and subband coefficients," in *Proc. SPIE*, vol. 2564, 1995, pp. 11-21.

[49] M. Antonini, M. Barlaud, P. Mathieu and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. Image Processing*, vol. 1, pp. 205-220, Apr. 1992.

[50] K. K. Parhi and T. Nishitani, "VLSI architectures for discrete wavelet transforms," *IEEE Trans. VLSI Systems*, pp.191-202, Jun., 1993.

[51] A. Grzeszczak, M. K. Mandal and S. Panchanathan, "VLSI implementation of discrete wavelet transform," *IEEE Trans. VLSI Systems*, vol. 4, pp.421-439, Dec. 1996.

[52] M. Vishwanath, R. M. Owens and M. J. Irwin, "VLSI architectures for the discrete wavelet transform," *IEEE Trans. Circuits and Systems*, vol. 42, pp. 305-316, May 1995.

[53] C. Yu and S. J. Chen, "Design of an efficient VLSI architecture for 2-D discrete wavelet transforms, *IEEE Trans. Consumer Electronic*, vol. 45, pp. 135-140, Feb. 1999.

[54] C. Chakrabarti, "A DWT-based encoder architecture for symmetrically extended images," in *Proc. IEEE Int. Symp. Circuits and Systems*, Orlando, U.S.A, pp. 123-126, May 1999.

[55] J. Ritter and P. Molitor, "A partitioned wavelet-based approach for image compression using FPGA's," in *Proc. IEEE Custom Integrated Circuits Conf.*, Orlando, pp. 547-550, May 2000.

[56] K. Haapala, P. Kolinummi, T. Hamalainen and J. Saarinen, "Parallel DSP implementation of wavelet transform in image compression," in *Proc. IEEE Int. Symp. Circuits and Systems*, Geneva, pp. 89-92, May 2000.

[57] G. K. Wallace, "The JPEG Still Picture Compression Standard," *Comm. ACM*, vol. 34, pp. 30-44, Apr. 1991.

[58] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg and D. J. LeGall, *MPEG video compression standard*, Chapman and Hall, 1996.

[59] A.M. Rassau, K. Eshraghian, T.C.B. Yu, H.N. Cheung, S.W. Lachowicz, W.A. Crossland and T.D. Wilkinson, "Smart pixel implementation of a 2-D parallel nucleic wavelet transform for mobile multimedia communications," in *Proc. Design, Automation and Test Conf.*, pp. 191-195, Feb. 1998.

[60] P. J. Ashenden, *The designer's guide to VHDL*, Morgan Kaufmann Publishers, Oct. 1999.

[61] S. Sjoholm and L. Lindh, *VHDLfor designers*, Prentice Hall, Dec. 1996.

[62] M. J. Smith, *Application-specific integrated circuits*, Addison Wesley Longman, Nov. 1996.

[63] D. Knapp, *Behavioral synthesis: digital system design using the Synopsys Behavioral Compiler*, Prentice Hall, May 1996.

[64] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys*, Kluwer Academic Publishers, Jan. 1995.

[65] E. H. Adelson and E. P. Simoncelli "Subband image coding with three-tap pyramids," in *Proc. Picture Coding Symposium*, 1990.

[66] *MTC-22000 CMOS 0.7μ standard cell family*, Alcatel Microelectronics.

[67] P. -Y. Cheng, J. Li and C. -C. J. Kuo, "Rate control for an embedded wavelet video coder," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, pp. 696-702, Aug. 1997.

[68] M. J. Ruf and J. W. Modestino, "Operational rate-distortion performance for joint source and channel coding of images," *IEEE Trans. Image Processing*, vol. 8, pp. 305-320, Mar. 1999.

[69] D. Chai and K. N. Ngan, "Face segmentation using skin color map in videophone applications," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, pp. 551-564, Jun. 1999.

[70] D. Chai and K. N. Ngan, "Content-based bit allocation and rate control for classical MC-DCT video coding systems," in *Proc. IEEE Int. Workshop Intelligent Signal Processing and Communication Systems*, Melbourne, Australia, vol. 2, pp. 601-605, Nov. 1998.

[71] Y. -Q. Zhang and S. Zafar, "Motion-compensated wavelet transform coding for color video compression," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 2, pp. 285-296, Sep. 1992.

[72] H. J. Kim and C. C. Li, "Lossless and lossy image compression using biorthogonal wavelet transforms with multiplierless operations," *IEEE Trans. Circuits and Systems: Analog and Digital Signal Processing*, vol. 45, pp. 1113-1118, Aug. 1998.

[73] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Processing*, vol. 9, no. 7, pp. 1158-1170, Jul. 2000.